PL3001 Quantum Computing
Project Report

# PRIORITIZING PATCH MANAGEMENT USING A QUANTUM-INSPIRED VERTEX COVER ALGORITHM

May 29, 2024

Alli Ajagbe
Ayush Sheth

# Contents

# 1 Abstract

Patch management is a fundamental aspect of cybersecurity, essential for mitigating vulnerabilities that attackers could exploit. Traditional methods for prioritizing patches primarily focus on the severity of individual vulnerabilities, often overlooking the interconnected nature of these vulnerabilities and their role in complex attack chains. This project investigates the application of quantum computing to enhance patch prioritization strategies by modeling the problem as a Minimum Vertex Cover (MVC) problem. Using the Classiq quantum computing framework, we represent vulnerability data as a bipartite graph and construct a dual graph to capture the interconnections between vulnerabilities. By applying the Quantum Approximate Optimization Algorithm (QAOA) to the MVC problem on the dual graph, we aim to identify a minimal set of critical vulnerabilities to patch. This approach seeks to disrupt potential attack paths more effectively than traditional methods, thereby enhancing the overall security posture. The results demonstrate the potential of quantum computing in transforming cybersecurity practices by providing a more holistic and interconnected view of vulnerabilities.

# 2 Introduction

The world is completely interconnected by networks today with the expansive reach of both technology and the internet. This has made cybersecurity a paramount need, as these networks are constantly under threat from malicious actors who exploit vulnerabilities in software systems to gain unauthorized access, steal sensitive data, or disrupt operations. Cybersecurity, as a field, has the aim to make these interconnected networks secure against such actors and their actions. This is where patch management steps in. Patch management is the process of identifying, acquiring, and deploying security patches to address exploitable vulnerabilities in a network, and is a critical component of any cybersecurity strategy. Although, given the uncountable, number of devices and subnetworks worldwide, and their constant exponential growth in numbers, traditional patch management approaches often struggle to keep up with both the ever-evolving threat landscape, and the sheer number of vulnerabilities that must be addressed.

This project aims to address this challenge by exploring a novel approach to prioritizing patch management using a method inspired by quantum computing. We are proposing utilizing the Minimum Vertex Cover problem to model a network system, and then leveraging the Quantum Approximate Optimization Algorithm (QAOA) to identify the most critical vulnerabilities to patch, thereby maximizing the effectiveness of limited resources.

## 2.1 Cybersecurity Threats Landscape

In recent years, the number of cybersecurity threats has grown exponentially due to the rapid penetration of technology into the global population. This trend is likely to continue. According to the IBM X-force Threat Intelligence index, there has been a **71 per cent** increase in the number of cyberattacks in 2024 compared to 2023. Moreover, for 2023, the number of cyberattacks was a **68 per cent** increase from 2022. The report is based on IBM's observations of over 150 billion daily security events worldwide[4]. This rise has been often attributed to a few major factors:

- The increased reliance on technology worldwide, where businesses and individuals are interconnected with an increasingly growing number of networks and software systems, providing a larger attack surface for malicious actors.

- The bettering sophistication of cyberattacks, with constant new and evolving techniques being adapted by malicious actors, making it harder for traditional security measures to keep up.

- The rise of ransomware attacks, which involve encrypting a victim's data and demanding a ransom in exchange for decryption and has caused significant financial losses and operational disruptions at a larger level.

These factors highlight the critical need for robust cybersecurity practices. Here, we can see how patch management would play a crucial role in addressing vulnerabilities in the software systems that malicious actors may exploit.

## 2.2 Traditional Patch Management and Challenges

As already understood, patch management is critical for cybersecurity. More imperative is the identification of the network components that require patching. However, with the ever-increasing volume of vulnerabilities that, prioritizing which ones to patch first proves difficult. Traditional methods rely on a combination of manual processes and vulnerability scoring systems, but they face several challenges in today's complex IT environments.

### 2.2.1 Vulnerability Scanning and Scoring Systems

Vulnerability scanning tools, such as the DAST (dynamic application security testing) tools, are employed to identify and assess potential vulnerabilities in the network system across all its components[5]. They identify vulnerabilities based on open-source data and assign each discovery a severity score based on a scale like the CVSS. The Common Vulnerability Scoring System (CVSS) is an industry-standard scoring system that assigns a severity score (0.0-10.0) based on exploitability, potential impact, and other characteristics. Scanners also consider the availability of the exploit in the system and prioritize the vulnerabilities based on open-source exploit code[6]. Affected systems are factored in and a higher priority is assigned to vulnerable systems that contain sensitive data or may directly impact critical components in the system infrastructure. Vulnerability scanning faces several challenges, the foremost being the sheer volume of vulnerabilities. Security teams not identifying critical vulnerabilities and instead spending resources on low-risk vulnerabilities also remains a challenge. Also, given how vulnerability scanning tools function, their reference points may lead them to identifying non-issues as vulnerabilities causing redundant resource allocation to patch them. Finally, the context of the organization cannot be an input to a scanning tool, and hence what may be an extreme high-risk vulnerability for a given organization's network may not even be an issue for another organization due to differences in the data sensitivity or network configuration.

### 2.2.2 Security Advisories and Vendor Bulletins[1]

While not explicitly a method, organizations are usually updated by their security software vendors on newly discovered vulnerabilities and potential spyware and malware that may affect their systems via security advisories and periodic bulletins. They usually detail vulnerabilities and their potential impacts, and the respective patch or mitigation strategy. Security teams at the organization may then prioritize and implement the patches to improve security according to their understanding.

The key challenge faced here is that the organization is completely at the mercy of the vendor and must rely on the vendor to release fixes on their own timeline. Critical vulnerabilities may remain exposed until the vendors approve a fix that may be implemented as a patch to improve security.

Such methods rely completely on the technical severity of the vulnerability, and do not factor in the real-world implications the vulnerability might have. Organizations need to categorize and prioritize vulnerabilities at their own behest based on the effect it might have on their network infrastructure and overall business outlook. For example, if a system has compromised financial data, it should receive priority patching over a less essential system with a same technical severity level. There is also the possibility of patching systems causing downtime for other software and applications that may interfere in the regular operations of the organization. All such issues need to be comprehensively examined while prioritizing patch deployment.

## 2.3 Motivation for the Quantum-Inspired Approach

The limitations of traditional approaches to patch management are evident, as in our prior discussion. Quantum computing is a rapidly evolving field, which has huge potential to revolutionize industries, including cybersecurity. While large-scale fault-tolerant quantum computers are still under development, there are various quantum-inspired algorithms that can be implemented using classical computers. One such algorithm is the Quantum Approximate Optimization Algorithm, developed by Farhi, Goldstone and Gutman in 2014[7], which is explained further in the report.

We explore the utility of the Quantum Approximate Optimization Algorithm, referred as QAOA forthwith, to tackle the challenge of prioritizing patch management. Our approach will utilize also the Minimum Vertex Cover problem. In graph theory, a vertex cover is a set of vertices in a graph that covers all edges, meaning every edge in the graph has at least one endpoint in the vertex cover set. The Minimum Vertex Cover problem aims to find the smallest possible vertex cover set for a given graph. For our purpose, we aim to model an interconnected network system that requires patching as a minimum vertex cover problem.

To elaborate, we can represent vulnerabilities and their dependencies within a network as a graph, where the nodes of the graph represent the assets of the network, which could either be the vulnerabilities in the network, or their dependencies. In this context, finding the minimum vertex cover of this vulnerability graph translates to identifying the smallest set of vulnerabilities that need to be patched to disrupt the largest number of potential attack chains. This alignment between the Minimum Vertex Cover problem and patch management is what made it a compelling candidate for optimization using a quantum-inspired method using QAOA.

### 2.4 Project Roadmap

The following sections will delve deeper into the technical details of our approach. We will explore the theoretical background of the Minimum Vertex Cover problem, the Quantum Approximate Optimization Algorithm (QAOA), and all other essential aspects pertinent to the project. We will then describe the specific methodology we have implemented, including the construction of the vulnerability graph and the integration and utilization of QAOA with the Classiq quantum computing framework. Finally, we will discuss the outcomes and evaluate the effectiveness of our proposed approach.

## 3 Methodology

### 3.1 Network Systems as Graph

This section leverages graphs to visualize how the problem is modelled. By representing the network system as nodes and edges, graphs not only concretize the system but also allow us to utilize powerful mathematical tools for optimizing patch management. This graphical approach simplifies understanding and problem-solving for complex network systems.

- **Nodes**: They represent assets, which are the individual components in the network system, such as servers, software applications, network devices (routers, firewalls), or operating systems.
- **Edges**: They represent the dependencies, which are the connections between the assets. A dependency between two assets allows a vulnerability in one to gain access to the other.

### 3.1.1 Vulnerabilities and Dependencies

**Vulnerabilities** are flaws or weaknesses in software or hardware, or their configuration, that can be exploited by a malicious actor. A compromised asset, which would be a node in the graph that is highlighted separate from the rest, would represent a vulnerability for our model.

**Dependencies** exist between two network components when they rely on each other to function according to their purpose. As mentioned earlier, the edges of the graphs connecting the assets or components depict these dependencies.

A vulnerability in an asset would mean the security of each asset dependent on it could be compromised. Further, if that keeps happening, it can easily snowball into a kill chain.

### 3.1.2 Kill Chains and Attack Graphs

In cybersecurity, a **kill chain** refers to the different stages an attacker goes through to compromise a network system. These stages typically involve reconnaissance, initial access, privilege escalation, lateral movement, installation, and command and control. Disrupting any stage in the kill chain can prevent a successful attack. This kill chain, for our network, can be represented as an **attack graph**. Attack graphs are a labelled transition system that models the malicious actor's movement in implementing a kill chain. To put it concisely, an attack graph depicts how and in what order an attacker exploits vulnerabilities to move through different components of a network to finally reach the asset of interest.

### 3.1.3 Vulnerability and Connectivity Dual Graphs

To understand a **vulnerability graph** better, it is necessary to understand **bipartite graphs**. A bipartite graph is where the nodes can be divided into two distinct sets, with the edges connecting nodes between both sets, but not within either set.

For our representation system, the first of the two sets could consist of the network assets, and the second set could consist of the network components that can be classified as vulnerabilities. Now these graphs edges are the dependencies, that we discussed earlier. Representing this graph as a bipartite system graph gives us our vulnerability graph.

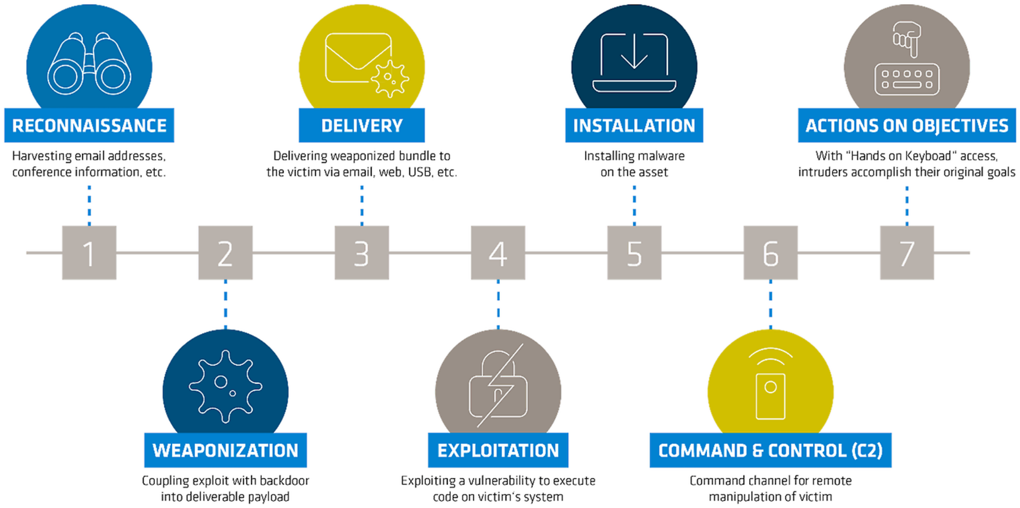

RECONNAISSANCE
Harvesting email addresses, conference information, etc.
DELIVERY
Delivering weaponized bundle to the victim via email, web, USB, etc.
INSTALLATION
Installing malware on the asset
ACTIONS ON OBJECTIVES
With "Hands on Keyboad" access, intruders accomplish their original goals
1
2
3
4
5
6
7
WEAPONIZATION
Coupling exploit with backdoor into deliverable payload
EXPLOITATION
Exploiting a vulnerability to execute code on victim's system
COMMAND & CONTROL (C2)
Command channel for remote manipulation of victim

Figure 1: Kill Chain Example[8]

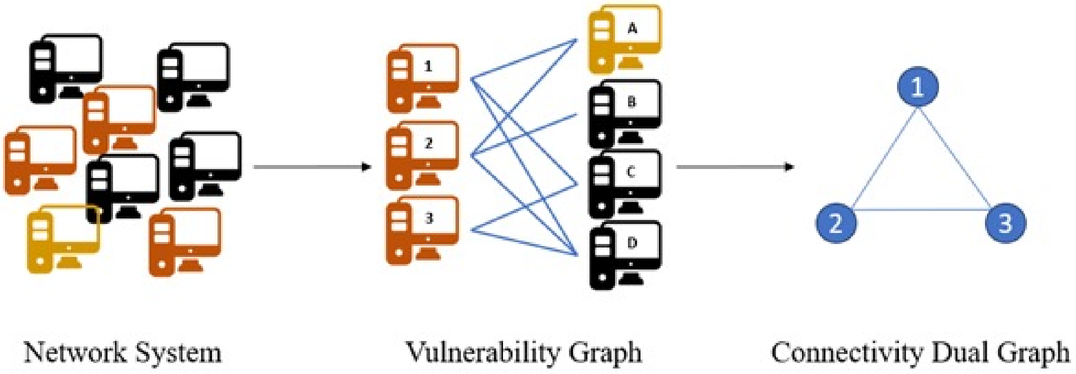
To add more context to the previously mentioned kill chain, a kill chain would consist of nodes in the vulnerabilities partition of our vulnerability bipartite graph, with at least one node in the assets partition linked as a dependency to any node of the vulnerabilities partition.

A **dual graph**, also called a line graph, according to graph theory is a new graph constructed using a prior graph. The nodes of the dual graph correspond to the edges of the prior graph, and two nodes in the dual graph are connected by an edge given their corresponding edges in the prior graph share a common node. For our problem scenario, we construct a **connectivity dual graph**, that makes it possible to identify clusters of vulnerabilities that are connected via a shared affected asset. We construct such a graph from our vulnerability graph by the following steps:

- Each node from the vulnerability set of the vulnerability graph is added as a node for the connectivity dual graph (nodes 1, 2, 3, in red).
- The assets (nodes A, B, C, D in black) associated with each node are iterated over.
- For each asset, the vulnerabilities that they are dependent on can be traced, ignoring the vulnerability we started with.
- For all such shared connections, an edge is created in the connectivity dual graph between vulnerabilities sharing a common asset.

- For better representation, vulnerability node pairs sharing more than one asset have a weighted edge between them in the connectivity dual graph, corresponding to the number of assets shared between the two.
  [In our problem although, we have not considered weighted edges for a simpler demonstration.]

- The process is repeated after removing the starting vulnerability node from the process.



Although, in the figure, we have a lesser number of nodes, it can be understood how detailed a connectivity dual graph may become upon extrapolating the process to larger real-life networks.

Through a connectivity dual graph, we have the required graph on which we can apply the **minimum vertex cover**. The connectivity dual graph, with the applied minimum vertex cover, allows us to map those vulnerabilities which need to be patched in order to disrupt attacks via kill chains. Each patched vulnerability in the connectivity dual graph will map to several dependent assets in the network, as can be understood by tracing back to the vulnerability graph. This shows the effectiveness of the approach, by minimizing the number of vulnerabilities patched to secure clusters of assets dependent on each patched vulnerability.

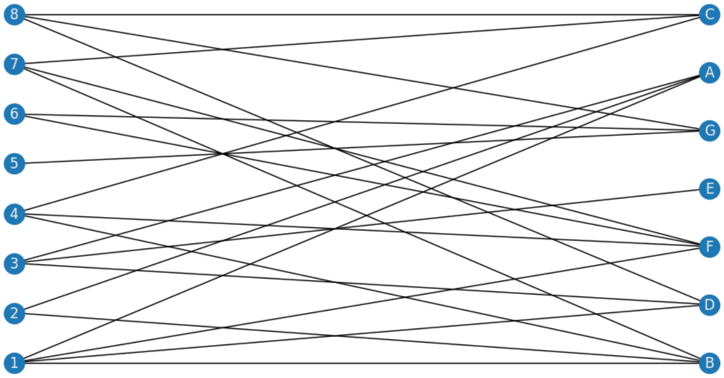


Figure 2: Bipartite Vulnerability Graph Example for a Larger Network[2]

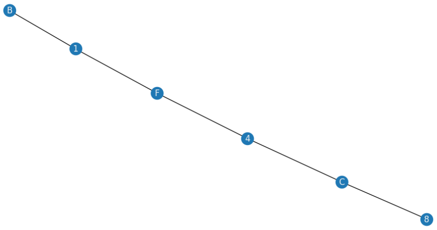


Figure 3: Implementable Kill Chain Example[2]

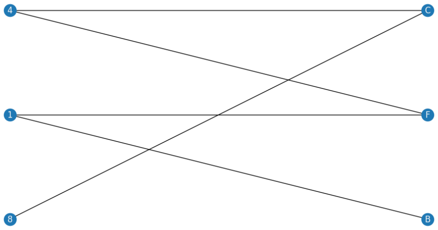


Figure 4: Kill Chain Vulnerability Graph[2]

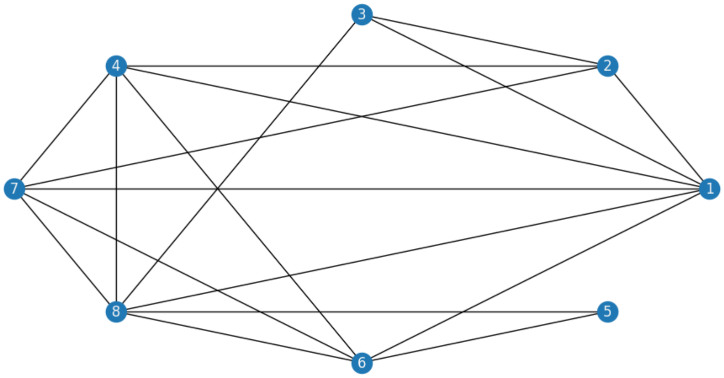


Figure 5: Connectivity Dual Graph for Network in Figure 2[2]

### 3.2 Minimum Vertex Cover Problem

A vertex cover in graph theory refers to a set of vertices that has a special property: every edge in the graph must have at least one of its endpoints in this set. In other words, a vertex cover touches every edge, ensuring that no edge is left uncovered. The mathematical formulation for the same is covered in a later section.

The minimum vertex cover problem is known to be an NP-complete problem, with traditional solution approaches having a time complexity $O(2^n)$ in the worst case, with non-dynamic programming approaches. [9] The complexity for vertex cover problem is similar to the MaxCut problem. Using QAOA, it has been shown that this complexity can be reduced to $O(poly(p))$ in a quasioptimal $p$-level execution for the MaxCut problem[10], which would imply a similar complexity for the minimum vertex cover we aim to solve for. The details of QAOA are covered in further sections.

In the context of our problem, the vertex cover would imply removing the nodes on the connectivity dual graph altogether, as the vulnerabilities would be patched. This would imply that our bipartite vulnerability graph will no longer be a graph, as both the sets of the graphs will become disconnected given how there will no longer be any vulnerabilities in the system. Further, to clarify, the minimum vertex cover would ensure the detachment of both sets in the vulnerability graph, but it would not imply that all nodes in the connectivity dual have been removed; it would rather mean that the minimum number of nodes to effectively secure the network will be patched, also meaning that the efficiency of patching will be high.

From our example with the network system and vulnerability graph, we can have various possible kill chains, such as those in Figure 1. The minimum vertex cover once implemented will remove all dependencies between the vulnerabilities and the assets that could allow for any kill chain, i.e., no asset would share two vulnerabilities.

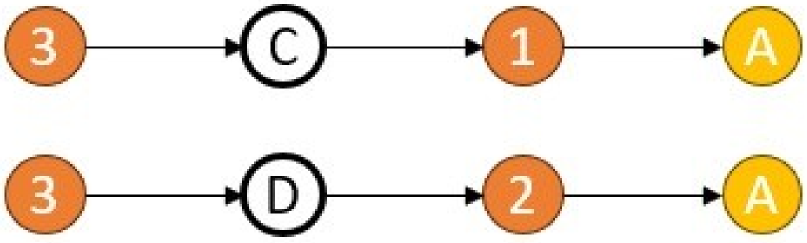


Figure 6: Possible KillChain

## 3.3 Understanding QAOA

The Quantum Approximate Optimization Algorithm (QAOA) is a hybrid quantum-classical algorithm designed to find approximate solutions to combinatorial optimization problems. This algorithm uses quantum circuits with classical optimization techniques, leveraging their respective strengths to enhance computational efficiency and solution quality.

QAOA is composed of two primary components: a parametric quantum circuit and a classical optimization segment. The parametric quantum circuit is constructed from quantum gates that encode the problem to be solved. These gates are parameterized, and the optimal values of these parameters are crucial for obtaining high-quality solutions. The execution of this circuit produces a quantum state, from which the most probable state is interpreted as a potential solution to the optimization problem.

The classical optimization segment is responsible for optimizing the parameters of the quantum gates. It employs classical algorithms such as Constrained Optimization BY Linear Approximations (COBYLA), Pyomo or gradient descent to iteratively adjust these parameters. During each iteration, the quantum gates are reapplied to the quantum state using the updated parameters, refining the solution progressively. The iterative process continues until the algorithm converges on a near-optimal solution. The quality of the solution is evaluated based on the Hamiltonian cost function, which the algorithm aims to minimize. The Hamiltonian cost function represents the objective function of the optimization problem, and its minimization indicates the proximity of the solution to the optimal configuration.

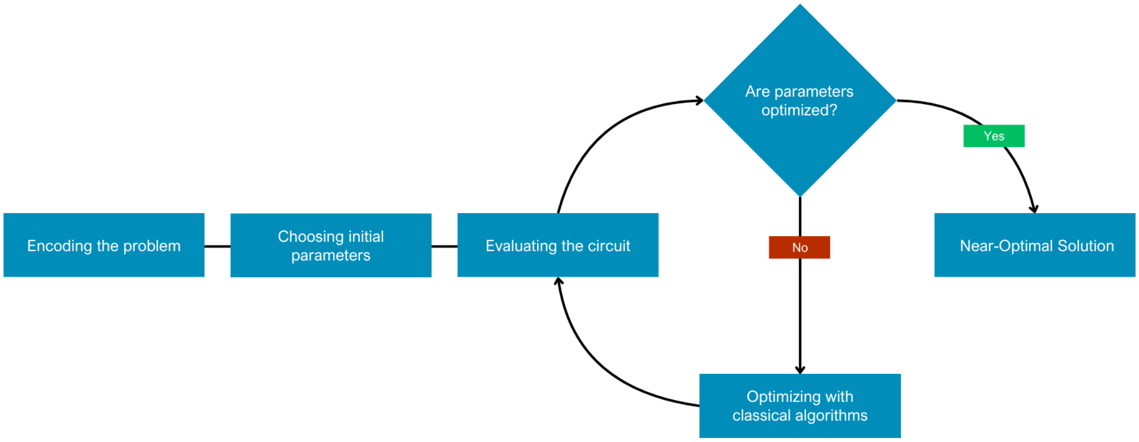

Are parameters optimized?
Yes
Encoding the problem
Choosing initial parameters
Evaluating the circuit
No
Near-Optimal Solution
Optimizing with classical algorithms

Figure 7: QAOA FlowChart[11]

### 3.3.1 Quantum Circuit

- Starting State: $|0\rangle^{\otimes n}$ where n is the number of qubits.
- State after Circuit is applied: $|\psi\rangle = \left|\psi(\bar{\gamma}, \bar{\beta})\right\rangle = \mathcal{U}(\bar{\gamma}, \bar{\beta})\,|0\rangle^{\otimes n}$ where $\bar{\gamma} = \gamma_1, \gamma_2, \ldots, \gamma_p, \bar{\beta} = \beta_1, \beta_2, \ldots, \beta_p$ are two sets of $p$ parameters.
- **QAOA Algorithm Circuit**: $\mathcal{U}(\bar{\gamma}, \bar{\beta}) = \mathcal{U}(\mathcal{H}_X, \beta_p)\mathcal{U}(\mathcal{H}_C, \gamma_p)\ldots\mathcal{U}(\mathcal{H}_X, \beta_1)\mathcal{U}(\mathcal{H}_C, \gamma_1)\mathcal{H}^{\otimes n}$ where:
  - $\mathcal{H}$: Hadamard gate
  - $\mathcal{U}(\mathcal{H}_X, \beta_j) = e^{-\iota\beta_j\mathcal{H}_X}$
  - $\mathcal{U}(\mathcal{H}_C, \gamma_j) = e^{-\iota\gamma_j\mathcal{H}_C}$
  - $\mathcal{H}_C$: Cost Hamiltonian which encodes the problem to be solved
  - $\mathcal{H}_X$: Mixing Hamiltonian defined as $\sum_{i=1}^{n} \sigma_i^x$
  - $\sigma_i^x$: Pauli X matrix applied to the *ith* qubit

The goal here is to find the optimal set of parameters $\bar{\gamma}^{opt}, \bar{\beta}^{opt}$ that minimizes the $\langle\psi|H_C|\psi\rangle$.

With the optimal parameters, $|\psi(\bar{\gamma}^{opt}, \bar{\beta}^{opt})\rangle$ is a superposition of basis states, and the state with the highest probability represents the solution to the problem. The optimization of these parameters is performed iteratively using classical optimization algorithms such as gradient descent, Pyomo (an open-source python optimization tool), or COBYLA, as previously mentioned.

The parameter $p$ in QAOA determines the depth of the quantum circuit. As $p$ approaches infinity, the algorithm is theoretically capable of finding a set of optimal parameters that yield an exact solution to the optimization problem. Conversely, when $p$ is finite, the algorithm provides an approximate solution. The trade-off between the depth of the quantum circuit and the computational resources required highlights the balance between solution accuracy and efficiency in QAOA[11].

## 3.4 QAOA for Minimum Vertex Cover (MVC)

For the minimum vertex cover problem, encoding is performed by representing the problem as a cost Hamiltonian, $\mathcal{H}_C$. This Hamiltonian maps the problem's variables to qubits and defines the cost function as an operator that acts on these qubits. The cost Hamiltonian, $\mathcal{H}_C$, encapsulates the constraints and objectives of the minimum vertex cover problem, ensuring that the quantum algorithm targets minimizing the corresponding cost function. By optimizing the parameters within the quantum circuit, QAOA aims to find the qubit configuration that represents the minimal vertex cover, thereby solving the problem. The Cost Hamiltonian is defined as shown below:

$$C(x) = \sum_{i \in V} x_i + \lambda \sum_{(i,j) \in E} (1 - x_i)(1 - x_j) \tag{1}$$

In the context of the minimum vertex cover problem, let $x \in \{0, 1\}^{|V|}$ be a binary variable vector representing whether a vertex is included in the vertex cover ($x_i = 1$) or not ($x_i = 0$). The parameter $\lambda > 1$ is a penalty parameter that controls the penalization strength for configurations that do not constitute a valid cover.

The first summation counts the size of the selected cover, and the second summation penalizes the configurations that do not cover all edges. Specifically:

- The term $\sum_{i \in V} x_i$ measures the number of vertices included in the vertex cover.
- The term $\lambda \sum_{(i,j) \in E} (1 - x_i)(1 - x_j)$ ensures that every edge $(i, j)$ in the graph is covered by at least one of its endpoints being part of the vertex cover. If both endpoints $i$ and $j$ are not in the cover ($x_i = 0$) and ($x_j = 0$), a penalty, $\lambda$, is added to the cost.

Minimizing *C(x)* ensures that we find a minimum vertex cover. The penalty parameter, $\lambda$, enforces that invalid configurations (those that do not cover all edges) are penalized heavily, driving the solution towards a valid and minimal vertex cover.

### 3.4.1 To QUBO

The Cost, *C(x)*, can be mapped to a Quadratic Unconstrained Binary Optimization (QUBO) problem which consists of finding a vector x such that the function, $f(x) = \sum_{i=1}^{n} h_i x_i + \sum_{i,j} J_{ij} x_i x_j$, is minimized. where:

- $x$ is a vector of n components ($x_i \in \pm 1$)
- $J_{ij}$ and $h_i$ are constants.

The function can then be encoded in a Cost Hamiltonian as below:

$$\mathcal{H}_C = \sum_{i=1}^{n} h_i \sigma_i + \sum_{i,j=1}^{n} J_{ij} \sigma_i \sigma_j \tag{2}$$

where $\sigma_i \in \{-1, 1\}$ are Ising variables and $h_i$ and $J_{i,j}$ represent the coefficients for the linear and quadratic coefficients, respectively. According to our MVC formulation where $x$ takes values in $\{$*0,1*$\}$, it is converted to $\{$*-1,1*$\}$ in order to retrieve the equivalent problem[12].

Now, correspondingly, to our MVC problem, we have the representation below:

$$\mathcal{H}_C = \sum_{i \in V} \frac{(1-\sigma_i^z)}{2} + \lambda \sum_{(i,j)\in E} \left(1 - \frac{(1-\sigma_i^z)}{2}\right)\left(1 - \frac{(1-\sigma_j^z)}{2}\right) \tag{3}$$

where $x_i$ is now represented as $\frac{1-\sigma_i^z}{2}$.

### 3.4.2 Decomposing the Circuit

The equations for the operations are hence:

$$\mathcal{U}(\mathcal{H}_X, \beta_j) = e^{-\iota\beta_j\mathcal{H}_X} = \Pi_k^n e^{-\iota\beta_j\sigma_k^x} = \Pi_k^n e^{-iC\sigma_k^x} \tag{4}$$

$$\mathcal{U}(\mathcal{H}_C, \gamma_j) = e^{-\iota\gamma_j\mathcal{H}_C} = \Pi_k^n e^{-\iota\gamma_j h_i\sigma_k^z}\Pi_{k,l} e^{-\iota\gamma_j J_{k,l}\sigma_k^z\sigma_l^z} = \Pi_k^n e^{-iC\sigma_k^z}\Pi_{k,l} e^{-iC\sigma_k^z\sigma_l^z} \tag{5}$$

- $e^{-iC\sigma_k^x}$ is the RX gate applied at the *kth* qubit.
- $e^{-iC\sigma_k^z}$ is the RZ gate applied at the kth qubit.
- $e^{-iC\sigma_k^z\sigma_l^z}$ is the RZZ gate applied at the *kth* and *lth* qubits.
- C is any constant.

With this, we are able to construct the QAOA circuit for the MVC problem [11].

### 3.4.3 Constructing the QAOA Circuit for the MVC Problem

Now that we understand how to encode the MVC problem into a parametric quantum circuit. To construct the QAOA circuit for the problem, we then follow the steps:

- **Initialization**: Preparing the initial quantum state as a superposition of all possible states.
- **Applying the Mixer Hamiltonian**: Implement the unitary operator $\mathcal{U}(\mathcal{H}_X, \beta_j)$ which applies the RX gates to each qubit. This operation is responsible for exploring the solution space.
- **Applying the Problem Hamiltonian**: Implement the unitary operator $\mathcal{U}(\mathcal{H}_C, \gamma_j)$ which applies the RZ and RZZ gates. This step encodes the cost function of the MVC problem into the quantum state.
- **Iterative Application**: Repeat the application of the Mixer Hamiltonian and Problem Hamiltonian for a specified number of iterations $p$. Each iteration involves updating the parameters $\beta_j$ and $\gamma_j$ using a classical algorithm.

- **Measurement**: Measure the final quantum state to obtain the solution corresponding to the vertex cover with the minimum cost.

The constructed QAOA circuit uses the specified quantum gates to iteratively optimize the parameters, gradually converging towards the minimum vertex cover. The effectiveness of this approach is evaluated based on the resulting solution's quality and computational efficiency.

## 4 Implementation with Classiq[2]

In this section, we detail the implementation of the approach to solving the Minimum Vertex Cover (MVC) problem using the Quantum Approximate Optimization Algorithm (QAOA) integrated with the Classiq quantum computing framework, as provided by Classiq Technologies itself[2]. The implementation involves several steps, including the construction of the optimization model, the preparation and execution of the quantum algorithm, and the visualization and evaluation of the results.

### 4.1 Graph Construction and MVC Model

We begin by constructing a bipartite graph, $\boldsymbol{B}$, using the *NetworkX* library. This graph consists of two sets of nodes: one representing assets and the other set representing the dependencies between them. The edges between these sets indicate the relationships that need to be covered.

```python
import networkx as nx
edge_dict = {
    1: ["A", "B", "D", "F"],
    2: ["A", "B"],
    3: ["A", "D", "E"],
    4: ["B", "C", "F"],
    5: ["G"],
    6: ["F", "G"],
    7: ["B", "C", "F"],
    8: ["C", "D", "G"],
}

B = nx.Graph()
B.add_nodes_from([1, 2, 3, 4, 5, 6, 7, 8], bipartite=0)
B.add_nodes_from(["A", "B", "C", "D", "E", "F", "G"], bipartite=1)
for u in range(1, 9):
    for v in edge_dict[u]:
        B.add_edge(u, v)
```

Next, we generate the dual graph, ***DG***, from the bipartite graph, ***B***. This dual graph represents the constraints of the MVC problem and serves as the input for our optimization model.

```python
def gen_dual(B_2, S=None):
B_2c = B_2.copy()
DualG = nx.Graph()
if not S:
    S, _ = nx.bipartite.sets(B_2c)
for s in S:
    DualG.add_node(s)
    for t1 in B_2c.neighbors(s):
        for t2 in B_2c.neighbors(t1):
            if t2 != s:
                DualG.add_edge(s, t2)
    B_2c.remove_node(s)
return DualG

S, _ = nx.bipartite.sets(B)
DG = gen_dual(B, S)
```

We then define the MVC optimization model using Pyomo, an open-source Python-based optimization modelling language. The model includes a binary variable for each node, a constraint to ensure full coverage of all edges and an objective function that minimizes the number of selected vertices.

```python
import pyomo.core as pyo

def mvc(graph: nx.Graph) -> pyo.ConcreteModel:
    model = pyo.ConcreteModel()
    model.x = pyo.Var(graph.nodes, domain=pyo.Binary)
    nodes = list(graph.nodes())

    @model.Constraint(graph.edges)
    def full_cover(model, i, j):
        return ((1 - model.x[i]) * (1 - model.x[j])) == 0

    def obj_expression(model):
        return sum(model.x.values())

    model.cost = pyo.Objective(rule=obj_expression, sense=pyo.
minimize)
    return model

mvc_model = mvc(DG)
print(mvc_model.pprint())
```

## 4.2 Quantum Algorithm Integration

We configure the QAOA algorithm to solve the MVC problem. This involves setting the number of QAOA layers and preparing the optimization model using Classiq's combinatorial optimization tools.

```
from classiq import construct_combinatorial_optimization_model, set_execution_preferences, write_qmod, show, synthesize, execute
from classiq.applications.combinatorial_optimization import OptimizerConfig, QAOAConfig

qaoa_config = QAOAConfig(num_layers=1)

def prepare_classiq(should_write):
    optimizer_config = OptimizerConfig(max_iteration=60, alpha_cvar=0.9)
    qmod = construct_combinatorial_optimization_model(
        pyo_model=mvc_model,
        qaoa_config=qaoa_config,
        optimizer_config=optimizer_config,
    )
    backend_preferences = ExecutionPreferences(
        backend_preferences=ClassiqBackendPreferences(backend_name="aer_simulator")
    )
    qmod = set_execution_preferences(qmod, backend_preferences)
    if should_write:
        write_qmod(qmod, "patching_mvc")
    return qmod

qmod = prepare_classiq(should_write=True)
```

## 4.3 Execution and Results

Authentication is done on the Classiq platform in order to get access to synthesize the quantum program. The quantum program is then executed on the ***aer_simulator*** backend, and the results are obtained.

```
classiq.authenticate()
qprog = synthesize(qmod)
show(qprog)

res = execute(qprog).result()
print(res)
```

The results are parsed to extract the Variational Quantum Eigensolver (VQE) solution, which includes the convergence graph. We visualize the convergence and analyze the solution to ensure it meets the MVC requirements.

```python
from classiq.execution import VQESolverResult
import matplotlib.pyplot as plt
import pandas as pd

vqe_result = VQESolverResult.parse_obj(res[0].value)
plt.figure(figsize=(20, 10))
vqe_result.convergence_graph
plt.show()

from classiq.applications.combinatorial_optimization import get_optimization_solution_from_pyo

solution = get_optimization_solution_from_pyo(
    mvc_model, vqe_result=vqe_result, penalty_energy=qaoa_config.penalty_energy
)
optimization_result = pd.DataFrame.from_records(solution)
optimization_result = optimization_result.sort_values(by="cost", ascending=True)
optimization_result.to_csv("patching_mvc_result.csv")

plt.figure(figsize=(20, 10))
optimization_result.hist("cost", weights=optimization_result["probability"])
plt.show()

best_solution = optimization_result.solution[optimization_result.cost.idxmin()]
print("Best Solution found:", best_solution)
```

## 4.4 Visualization

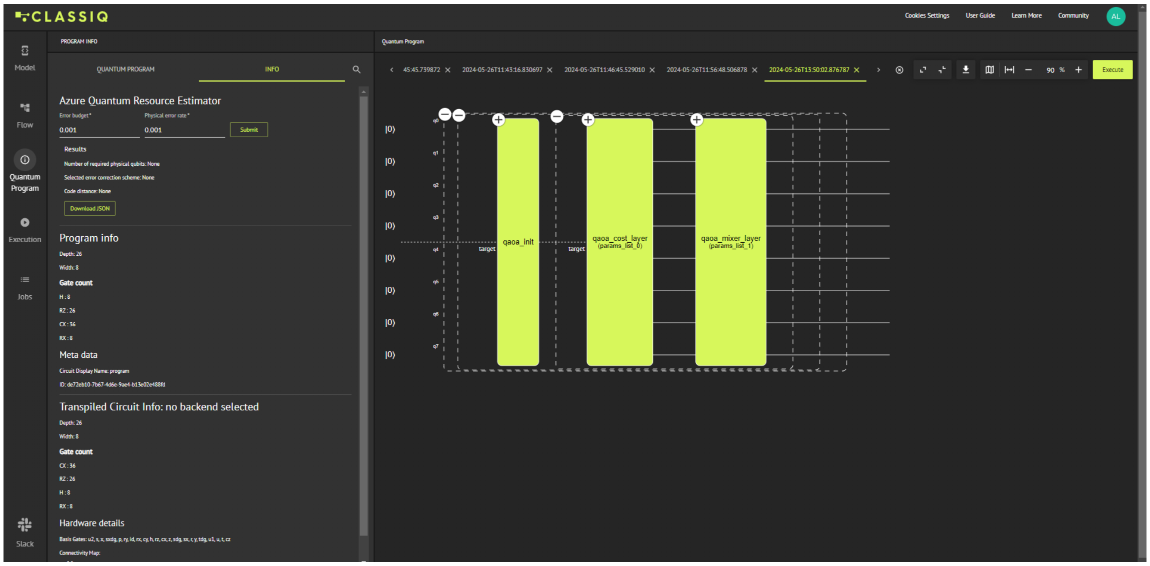
Finally, we visualize the graph and the selected vertex cover to demonstrate the effectiveness of the solution. The nodes and edges that are part of the vertex cover are highlighted.

```python
def draw_solution(graph: nx.Graph, solution: list):
    solution_nodes = [v for v in graph.nodes if solution[v - 1]]
    solution_edges = [
        (u, v) for u, v in graph.edges if u in solution_nodes or v in solution_nodes
    ]
    nx.draw_kamada_kawai(graph, with_labels=True)
    nx.draw_kamada_kawai(
        graph,
        nodelist=solution_nodes,
        edgelist=solution_edges,
        node_color="r",
        edge_color="y",
    )

plt.figure(figsize=(20, 10))
draw_solution(DG, best_solution)
plt.show()

check_B = B.copy()
vc2 = [v for v in DG.nodes if best_solution[v - 1]]
for v in vc2:
    check_B.remove_node(v)

plt.figure(figsize=(20, 10))
nx.draw(
    check_B,
    pos=nx.bipartite_layout(check_B, S),
    with_labels=True,
    font_color="whitesmoke",
)
plt.show()
```

This concludes the implementation of the Minimum Vertex Cover problem using Classiq.

# 5 Results[3]

Upon authentication, we execute the quantum circuit on the Classiq Web IDE to obtain the energy convergence results as shown in the Classiq images below. As seen in Figure 8,



Azure Quantum Resource Estimator

Error budget*: 0.001

Physical error rate*: 0.001

Results

Number of required physical qubits: None

Selected error correction scheme: None

Code distance: None

Program info

Depth: 26

Width: 8

**Gate count**

H : 8

RZ : 26

CX : 36

RX : 8

Meta data

Circuit Display Name: program

ID: de72eb10-7b67-4d6e-9ae4-b13e02e488fd

Transpiled Circuit Info: no backend selected

Depth: 26

Width: 8

**Gate count**

CX : 36

RZ : 26

H : 8

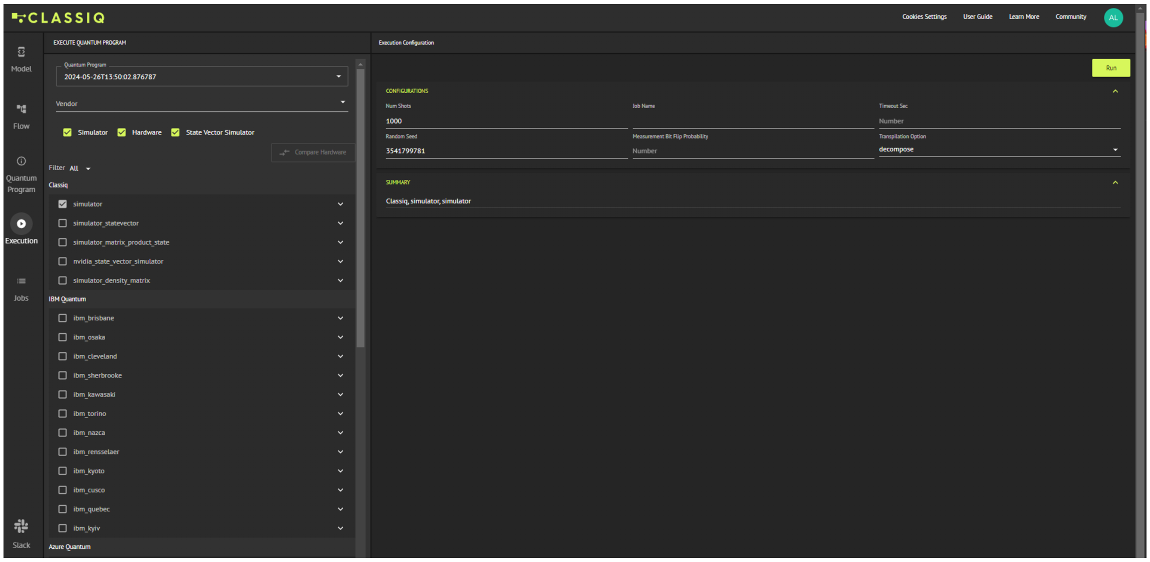
RX : 8

Hardware details

Basis Gates: u2, s, x, sxdg, p, ry, id, rx, cy, h, rz, cx, z, sdg, sx, r, y, tdg, u1, u, t, cz

Connectivity Map:

Figure 8: The Circuit Ready for Execution



Quantum Program: 2024-05-26T13:50:02.876787

Vendor

☑ Simulator ☑ Hardware ☑ State Vector Simulator

Filter All

Classiq

- ☑ simulator
- □ simulator_statevector
- □ simulator_matrix_product_state
- □ nvidia_state_vector_simulator
- □ simulator_density_matrix

IBM Quantum

- □ ibm_brisbane
- □ ibm_osaka
- □ ibm_cleveland
- □ ibm_sherbrooke
- □ ibm_kawasaki
- □ ibm_torino
- □ ibm_nazca
- □ ibm_rensselaer
- □ ibm_kyoto
- □ ibm_cusco
- □ ibm_quebec
- □ ibm_kyiv

Azure Quantum

Num Shots: 1000

Job Name:

Timeout Sec: Number

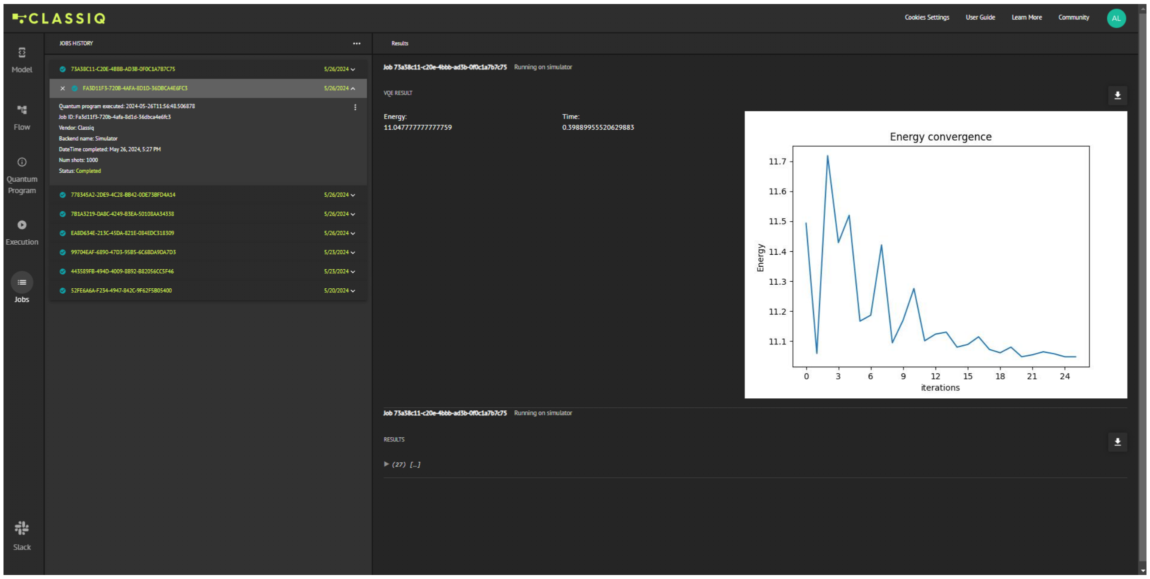
Random Seed: 3541799781

Measurement Bit Flip Probability: Number

Transpilation Option: decompose

Classiq, simulator, simulator

with an increase in the number of iterations, we approach a solution with a minimal cost



JOBS HISTORY

73A38C11-C20E-4BBB-AD3B-0F0C1A7B7C75 5/26/2024

FA3D11F3-720B-4AFA-8D1D-36DBCA4E6FC3 5/26/2024

Quantum program executed: 2024-05-26T11:56:48.506878
Job ID: Fa3d11f3-720b-4afa-8d1d-36dbca4e6fc3
Vendor: Classiq
Backend name: Simulator
DateTime completed: May 26, 2024, 5:27 PM
Num shots: 1000
Status: Completed

778345A2-2DE9-4C28-BB42-0DE73BFD4A14 5/26/2024

7B1A3219-DA8C-4249-B3EA-50108AA34338 5/26/2024

EA8D634E-213C-45DA-821E-084EDC318309 5/26/2024

99704EAF-6890-47D3-95B5-6C6BDA9DA7D3 5/23/2024

443589FB-494D-4009-8B92-B82056CC5F46 5/23/2024

52FE6A6A-F234-4947-842C-9F62F5B05400 5/20/2024

Results

Job 73a38c11-c20e-4bbb-ad3b-0f0c1a7b7c75 Running on simulator

VQE RESULT

Energy:
11.047777777777759

Time:
0.39889955520629883

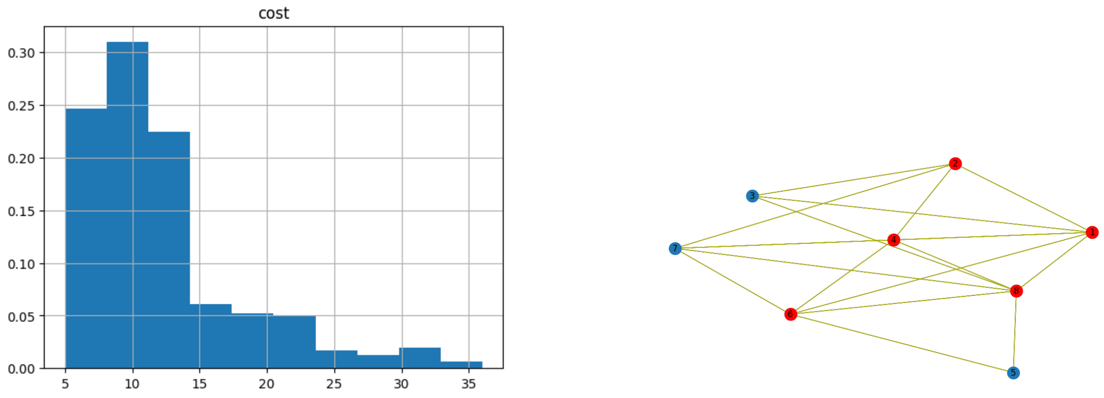
Job 73a38c11-c20e-4bbb-ad3b-0f0c1a7b7c75 Running on simulator

RESULTS

▶ (27) [...]

Figure 9: Result Obtained with Energy Convergence Graph

- we closely approach optimality as the number of iterations increases. We can also see correspondingly, that the solution found is the one with the highest probability, as shown in Figure 10a. Finally, the minimum vertex cover is highlighted over the graph in Figure 5, as shown in Figure 10b.



(a) Histogram with Solution Probabilities

(b) Final MVC Solution

Figure 10: Histogram and Final MVC Solution

## 5.1 Tuning the Architecture Parameters

In our initial implementation, we observed that the solution found with the highest probability was not always the one with the lowest cost. To address this issue, we explored the tuning of the architecture parameters to improve the accuracy and efficiency of the QAOA algorithm.

### 5.1.1 Increasing Number of Layers

Here, we increase the number of layers to 2 but retain the number of iterations at 60. With this, we see that a solution with a slightly lower cost is returned with the highest

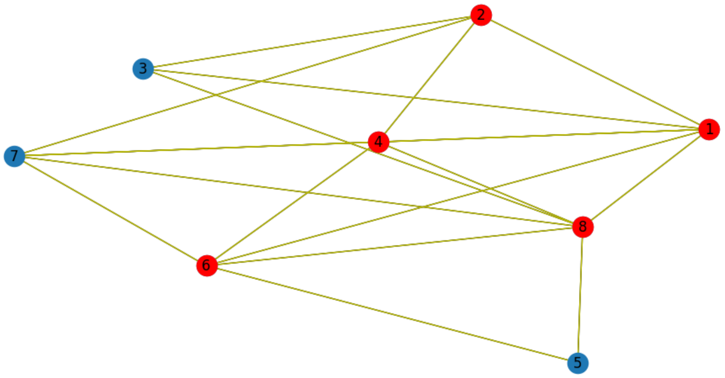
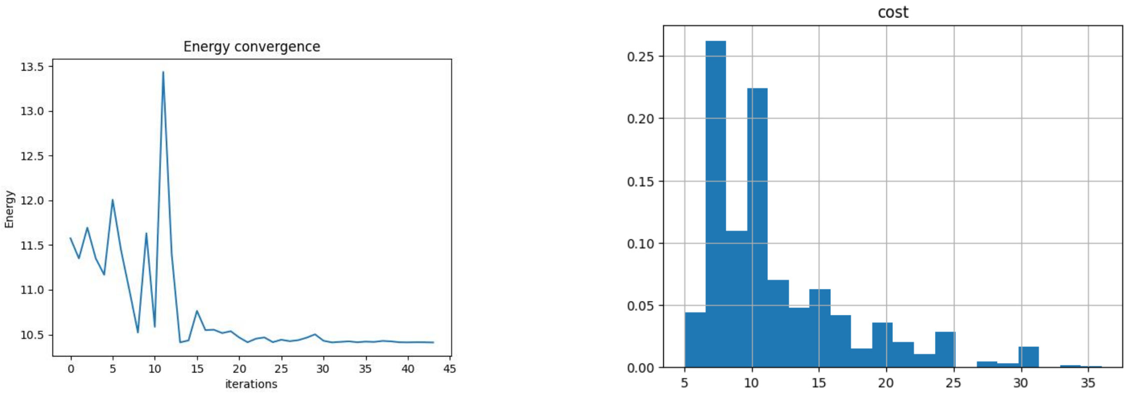


Figure 11: Minimum Vertex Cover Solution Found



(a) Energy Convergence Graph

(b) Histogram with Solution Probabilities

Figure 12: Energy Convergence Graph and Cost Histogram

probability. This infers that increasing the number of QAOA layers can help in refining the solution for the MVC problem.

### 5.1.2 Increasing Number of Layers & Iterations

Here, we increase the number of layers to 3 and the number of iterations to 100. With this, we see an even better solution with the minimum cost and maximum probability. One important thing to note, however, is that an increase in the number of layers for the QAOA algorithm further increases the algorithm's time complexity.

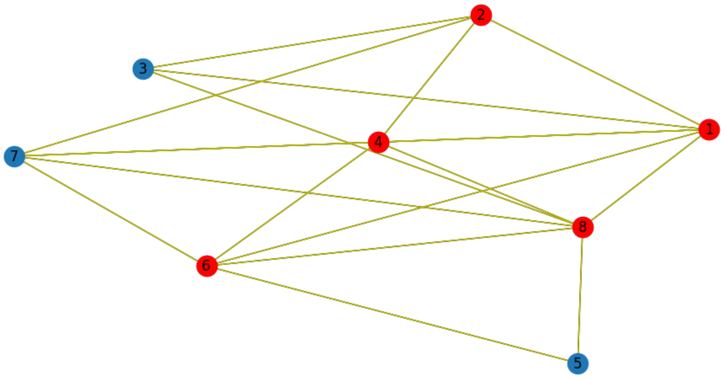
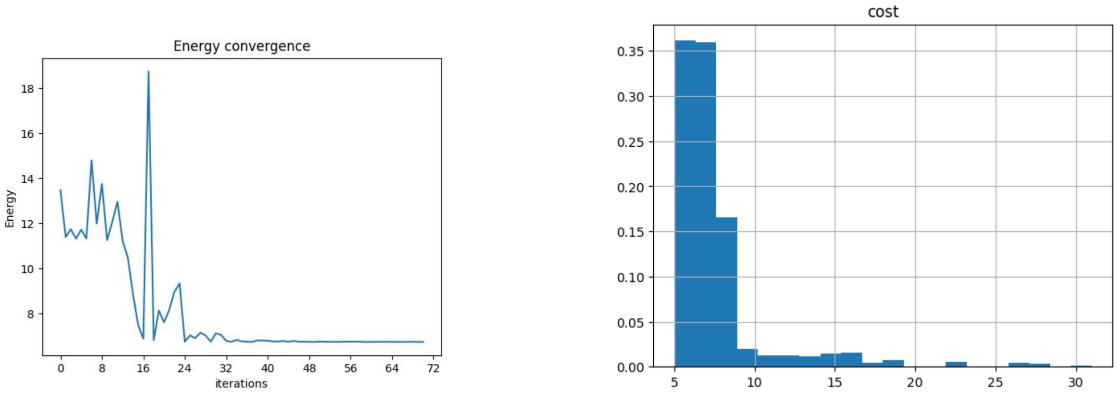


Figure 13: Minimum Vertex Cover Solution Found



(a) Energy Convergence Graph

(b) Histogram with Solution Probabilities

Figure 14: Energy Convergence Graph and Cost Histogram

### 5.2 Github

This is the Github Repo Link[3]. It contains supplementary materials to the project, such as codes and elaborate explanations for various topics.

## 6 Future Direction and Conclusion

There are more than a few directions that this project could take when extended and worked upon further. The first one of those is applying this methodology to larger scale network systems. While we have worked with a few nodes in our simulation, this could be extended to much a much larger number of nodes given adequate computation power. Although it is difficult to know how to translate such actual physical systems to a computable computational model.

It is necessary again to enunciate how this is a quick and efficient method to prioritize the vulnerabilities that require patching; and given how the technological world is evolving and expanding, quicker and more efficient cybersecurity patching methods are imperative. If this particular methodology can be applied to real world network systems to decide patching priorities, it could save a lot of resources and time for organizations. Again, we are not utilizing actual quantum computers to run the quantum simulation of the method, and we will not be able to do so until there are efficient, commercially available, and actually feasible quantum computers available for use.

Another direction this project could take in the future is the application and utilization of this method in networks enabled by internet-of-things (IoT). Given such networks can run quantum computing software, we can utilize the method to diagnose these network systems for various problems, such as connectivity for a wireless connectivity network. Generally, the method can be extrapolated to any problem that requires particular nodes to be patched, literally or figuratively, and can be represented as graph systems. Demonstrating solving of such problems could also be a vein in which the project could be taken forward.

We have demonstrated how prioritizing patching can be solved using a graph-based, quantum-inspired approach to speed up the process with a reduced time complexity. While quantum computers are not available at hand for consumers, the applications of such methods highlights the transformative potential quantum computing carries. Given how quantum computing, and quantum computers as such, are quicker than traditional computational technology, the time complexity for such problems can be brought down drastically. Especially for real world optimization problems, the utility of quantum principles via computing can pave roads to effectively address these problems, saving both time and resources. With the ever-evolving technological landscape, evolution and innovation in methodologies will certainly aid and shape the landscape and how it addresses various problems.

## References

[1] Security advisories. Accessed on 2024-05-26. [Online]. Available: https://sec.cloudapps.cisco.com/security/center/publicationListing.x

[2] ClassIQ, “THE VERTEX COVER PROBLEM WITH APPLICATION IN CYBER SECURITY - ClassIQ,” accessed on 2024-05-25. [Online]. Available: https://docs.classiq.io/latest/explore/applications/cybersecurity/patching_management/patching_managment/

[3] AlliAjagbe and AyushSheth, “Github - alliajagbe/patch-mgt: Prioritizing patch management for cybersecurity using quantum-inspired vertex cover algorithm,” https://github.com/alliajagbe/patch-mgt, accessed: 2024-05-26.

[4] “Ibm security x-force threat intelligence index 2024,” https://www.ibm.com/reports/threat-intelligence, accessed: 2024-05-25.

[5] Vulnerability scanning tools. OWASP Foundation. Accessed on 2024-05-26. [Online]. Available: https://owasp.org/www-community/Vulnerability_Scanning_Tools

[6] NVD - Vulnerability Metrics. Accessed on 2024-05-26. [Online]. Available: https://nvd.nist.gov/vuln-metrics/cvss

[7] E. Farhi, J. Goldstone, and S. Gutmann, “A quantum approximate optimization algorithm,” *arXiv*, 2014. [Online]. Available: https://doi.org/10.48550/arxiv.1411.4028

[8] Cyber kill chain. Accessed on 2024-05-26. [Online]. Available: https://www.adesso.de/en/news/blog/cyber-kill-chain.jsp

[9] Solving the vertex cover problem. Accessed on 2024-05-29. [Online]. Available: https://iq.opengenus.org/vertex-cover-problem/

[10] L. Zhou, S.-T. Wang, S. Choi, H. Pichler, and M. D. Lukin, “Quantum approximate optimization algorithm: Performance, mechanism, and implementation on near-term devices,” *Physical Review X*, 2020. [Online]. Available: https://doi.org/10.1103/PhysRevX.10.021067

[11] EntropicaLabs, “What is the qaoa?” https://openqaoa.entropicalabs.com/what-is-the-qaoa/#description-of-the-algorithm, 2024, accessed: 2024-05-25.

[12] Entropicalabs, “What is a qubo,” https://openqaoa.entropicalabs.com/problems/what-is-a-qubo/#how-to-write-a-qubo, 2024, accessed: 2024-05-2025.