CLASSIQ

QUANTUM OPTIMIZATION WITH CLASSIQ

Application note | August 2021

Introduction

Optimization problems are found in numerous industries. Here are just a few examples:

- **Portfolio optimization**. A money manager has a certain amount of money to invest. What is the optimal allocation of these funds?
- **Route optimization**. A FedEx truck needs to deliver 50 packages. What is the optimal route to deliver the packages?
- **Airport security**. An airport wishes to install security cameras to watch over all corners in the airport. What is the optimal arrangement of cameras that achieves this?

Each optimization has some definition of what constitutes an optimal solution. For instance, the money manager might seek the highest return with the lowest risk. The FedEx truck might want to complete the deliveries in the minimum amount of time. The airport might wish to cover all corridors with the smallest number of cameras.

The definition of what constitutes an optimal solution might be different from one company to another. Instead of the shortest delivery time, another company might choose the lowest-cost route, considering fuel and toll charges. Another might choose the route that generates the lowest carbon footprint.

Each optimization might also be done under a different set of constraints. The money manager, for instance, can't invest more than 5% of her portfolio in any single stock. The delivery truck might not be able to travel on certain residential roads before 8 AM, and so forth. An optimization problem that has constraints is sometimes called a "Constraint Optimization Problem"

It's easy to see how optimization problems can become exceptionally complex. The FedEx truck with 50 stops has about 30 vigintillion (that's 3×10^{64}) possible options. Even if the optimization can be completed in a reasonable amount of time, it might need to be redone at a moment's notice: There is a major traffic jam on the route, the price of an asset has dropped making it more attractive for purchase, etc.

But when problems are hard, solving them could provide a big reward. A logistics company that saves 15% on fuel costs because of optimized routing can increase its profits or grow its market share. An optimal portfolio is good for the customer, the portfolio manager, and her employer.

The difficulty of solving these problems and the payoff of getting the best answers are key reasons that companies are considering quantum computing. To put this in a financial context, the Boston Consulting Group recently estimated that there is between $110-210B of value that can be unlocked in solving optimization problems using quantum computers.

Integer Linear Program Optimization Problems

Integer Linear Program (ILP) is a class of optimization problems. The problem statement is to find an optimal choice by combining objects from a finite set. One such example is the Knapsack problem.

Knapsack. The story: a boy wishes to go on a hike with a knapsack of a known size. He can select which items to pack in the knapsack. Each item has a value and a size associated with it. What is the best combination that maximizes the value of the items in the knapsack while making sure that the aggregate size of the items does not exceed the size of the knapsack?

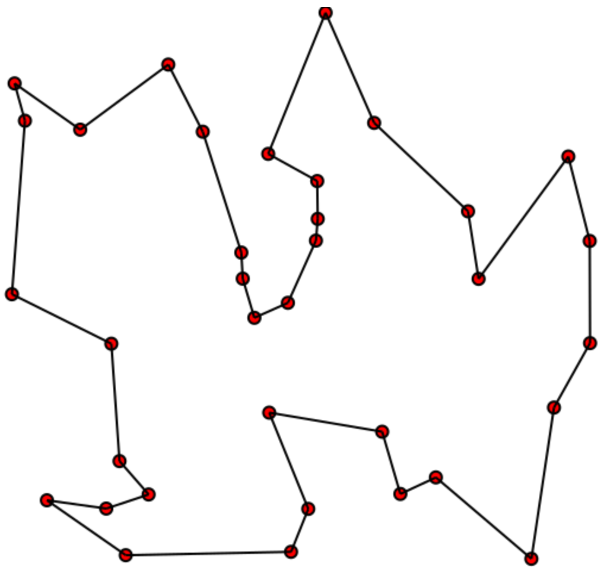
A production scheduling optimization problem might be considered a knapsack problem: pick the right production schedule ("right" could be, for instance, the lowest inventory holding costs while avoiding stock-outs) without exceeding the available production capacity.

Additional problems that often fall under the ILP category include network flows, assignment problems, and more.

Graph Optimization Problems

Another common class of optimizations problems is graph optimization problems. One such example is the traveling salesperson (TSP) problem:

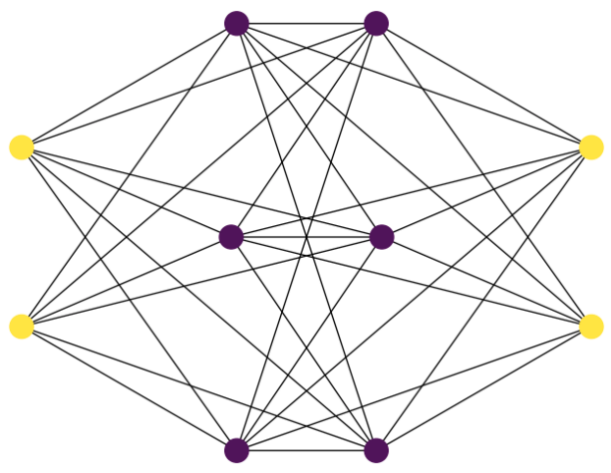
Traveling salesperson. The story: a salesperson needs to visit a certain set of customers in a day, perhaps returning to his home at the end of the day. What is the optimal route she should take? Obviously, the route optimization problem described earlier is such a problem. For instance, the image on the right[1] shows the shortest possible loop that connects all red dots.



[1] By Xypron - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=10645665

Another example is the vertex cover problem:

A vertex cover of a graph is the set of vertices that include at least one vertex (endpoint of every edge) of the graph. This would correspond to the airport security problem we described above. The airport would be described as a connected graph where each edge is a corridor. We can place cameras at either end of the corridor (the vertex of the graph). Thus, finding the vertex cover would provide us with the minimal set of vertices (or cameras) that are needed to cover all the corners of the airport.



For instance, if the airport is described in the graph on the right, if we place four cameras in the corners painted in yellow, we would be able to cover the largest possible number of corners. If we wanted to cover additional corners, we would need to add cameras.

Enter Quantum Computing

Two fundamental properties of quantum - superposition and entanglement - allow quantum computers to examine numerous options simultaneously. Once an optimization problem can be translated into a quantum circuit, quantum computers can look for a solution. To determine which solution is better and which is worse, quantum scientists create an “Oracle” which assigns a quantitative score to each potential solution. The circuit then works to find a solution with the lowest score.

The Classiq Approach

When we created our optimization libraries, our goal was to allow those that are not experts in quantum computing to use quantum algorithms for their optimization problems.

To solve these optimization problems, we used the QAOA (Quantum Approximate Optimization Algorithm) algorithm, one of the main algorithms for NISQ-era quantum computing. We will not describe the QAOA here, but good places to read about it are here, here and here.

Here are a few examples of solving optimization problems with Classiq:

Graph Optimization: Maximal Independent Set

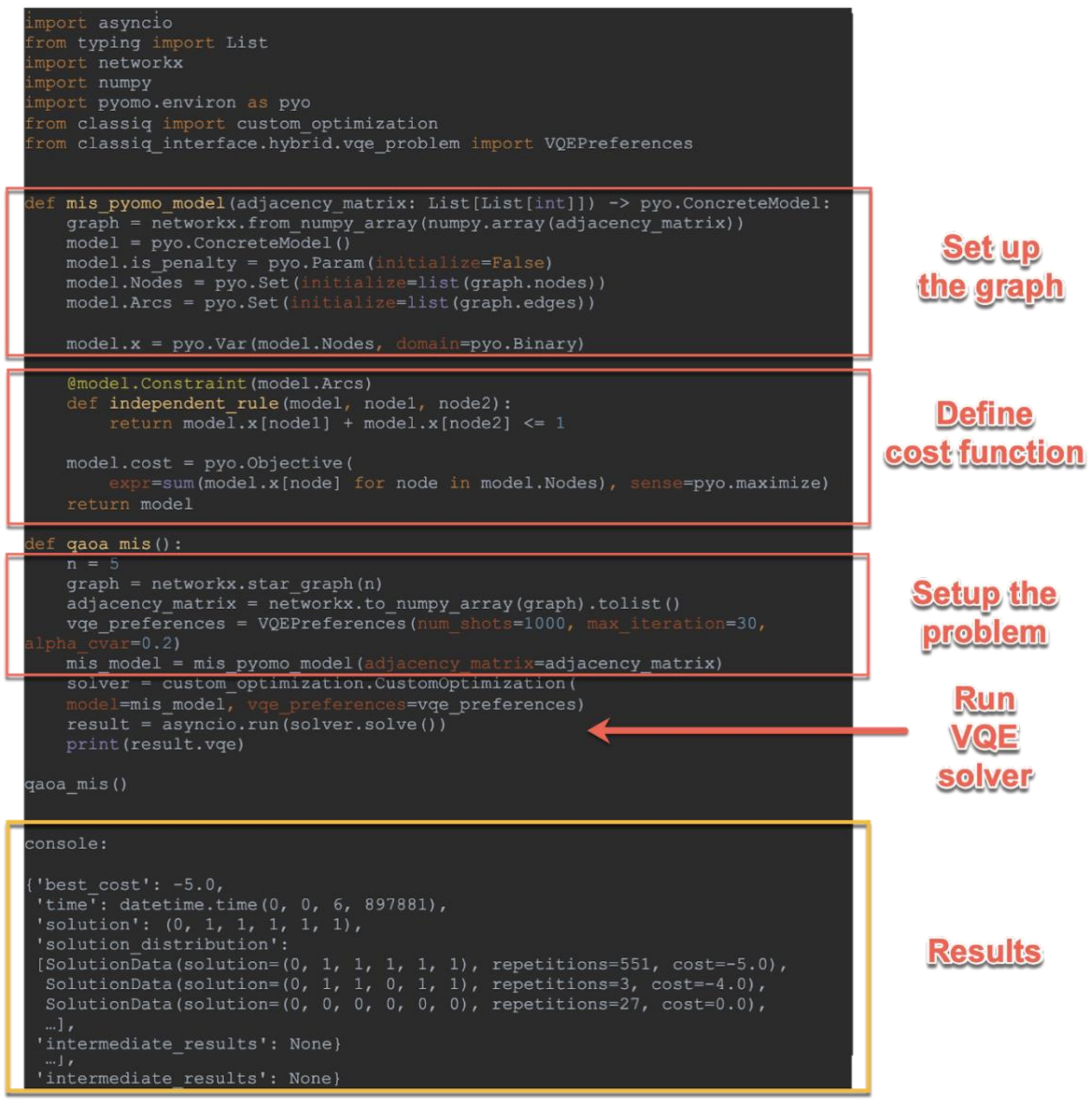
The Maximal Independent Set (MIS) is the largest set of vertices such that no two vertices in the set are connected by an edge (and therefore not deemed to be independent).

Here is an example that can be modeled as a maximal independent set:

> *A college is holding a day of performing arts concerts for its students. Each student chooses events of interest from the full list of concerts. The college wants to schedule the events in 1-hour time slots such that every student can visit all their chosen concerts without conflict.*

A possible embedding of this problem into MIS problem is the following: Construct a graph in which each concert is represented by a vertex, and two vertices are connected with an edge if at least one student chose both events

Here is the Classiq code that implements finding such MIS on various graphs:



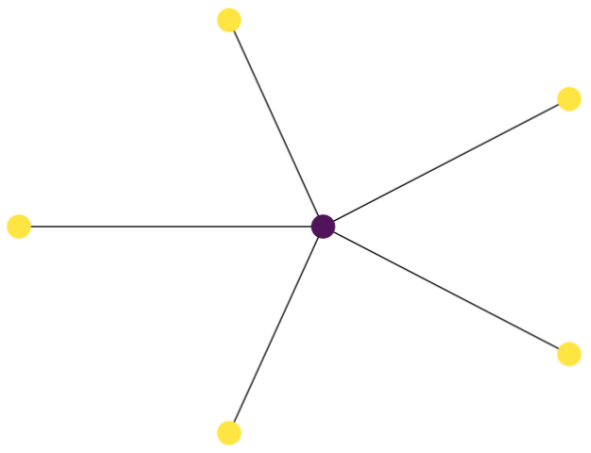
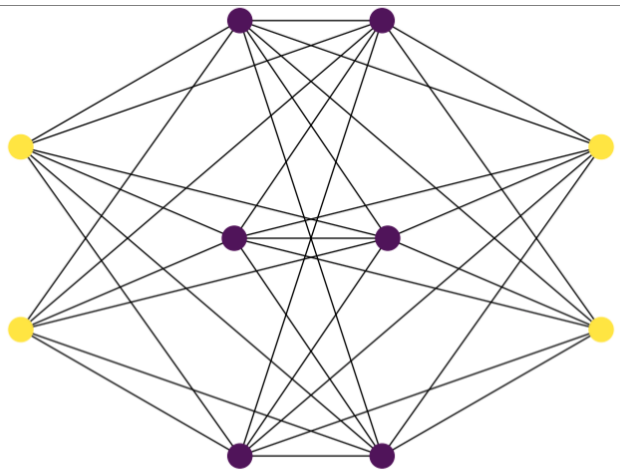
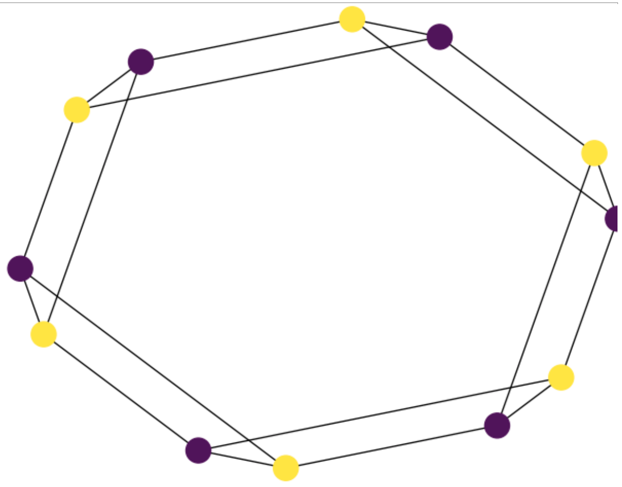
```
import asyncio
from typing import List
import networkx
import numpy
import pyomo.environ as pyo
from classiq import custom_optimization
from classiq_interface.hybrid.vqe_problem import VQEPreferences

def mis_pyomo_model(adjacency_matrix: List[List[int]]) -> pyo.ConcreteModel:
    graph = networkx.from_numpy_array(numpy.array(adjacency_matrix))
    model = pyo.ConcreteModel()
    model.is_penalty = pyo.Param(initialize=False)
    model.Nodes = pyo.Set(initialize=list(graph.nodes))
    model.Arcs = pyo.Set(initialize=list(graph.edges))

    model.x = pyo.Var(model.Nodes, domain=pyo.Binary)

    @model.Constraint(model.Arcs)
    def independent_rule(model, node1, node2):
        return model.x[node1] + model.x[node2] <= 1

    model.cost = pyo.Objective(
        expr=sum(model.x[node] for node in model.Nodes), sense=pyo.maximize)
    return model

def qaoa_mis():
    n = 5
    graph = networkx.star_graph(n)
    adjacency_matrix = networkx.to_numpy_array(graph).tolist()
    vqe_preferences = VQEPreferences(num_shots=1000, max_iteration=30,
alpha_cvar=0.2)
    mis_model = mis_pyomo_model(adjacency_matrix=adjacency_matrix)
    solver = custom_optimization.CustomOptimization(
    model=mis_model, vqe_preferences=vqe_preferences)
    result = asyncio.run(solver.solve())
    print(result.vqe)

qaoa_mis()
```

```
console:

{'best_cost': -5.0,
 'time': datetime.time(0, 0, 6, 897881),
 'solution': (0, 1, 1, 1, 1, 1),
 'solution_distribution':
 [SolutionData(solution=(0, 1, 1, 1, 1, 1), repetitions=551, cost=-5.0),
  SolutionData(solution=(0, 1, 1, 0, 1, 1), repetitions=3, cost=-4.0),
  SolutionData(solution=(0, 0, 0, 0, 0, 0), repetitions=27, cost=0.0),
  …],
 'intermediate_results': None}
  …],
 'intermediate_results': None}
```

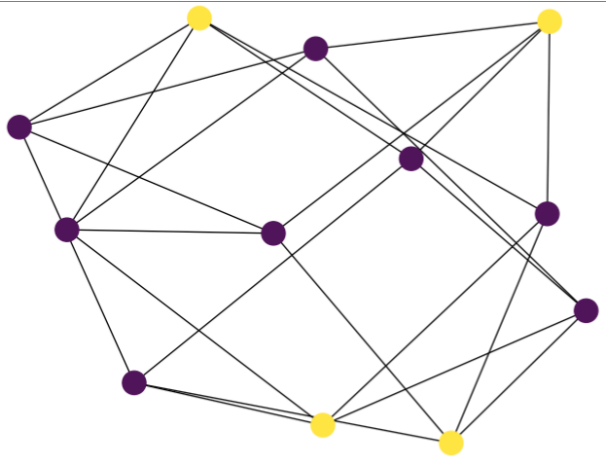
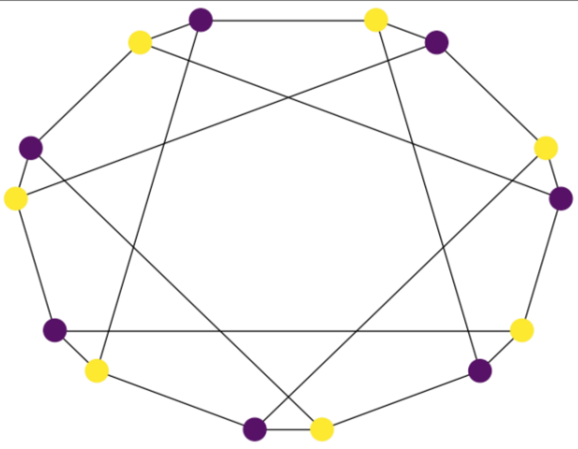
The key steps are as follows:

- Set up the graph. In this code sample, we use the networkx package to create a star topology with 5 nodes. Below, we show several runs from this package, including star, Turan, LCF, and more.
- Calculate the adjacency matrix, a square matrix showing whether pairs of vertices are adjacent or not
- Set up the VQE preferences: number of shots, maximum iterations in each shot
- Create the cost function for each graph. As written, the function discourages (gives a higher value) for graphs where the assigned variables for two adjacent vertices are 1
- Run the VQE optimization using the Maximum Independent Set model
- Report the results. We see that in this case, the most common solution is the one that the maximum independent set does not include the center (the first index in the graph) but does include all the nodes.

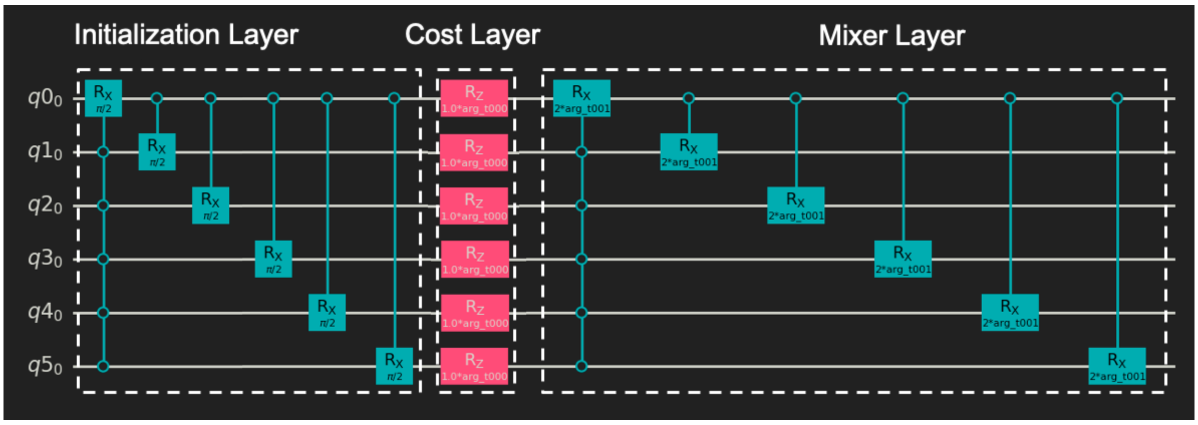
Here are the solutions for various graph types:

Graph type (using networkx library)	Minimum Independent Set solution[2]
star_graph(5). The solution shows a set of five selected vertices (yellow dots) arranged in such a way that no two are adjacent to each other and that adding any additional yellow vertex to the set (in this case, in the middle of the star) would violate the "no adjacency" rule.	
turan_graph(10, 3)	
LCF_graph(12, [3, -3], 5)	

[2] We used the networkx library to create these graphs from the solutions of the quantum circuit

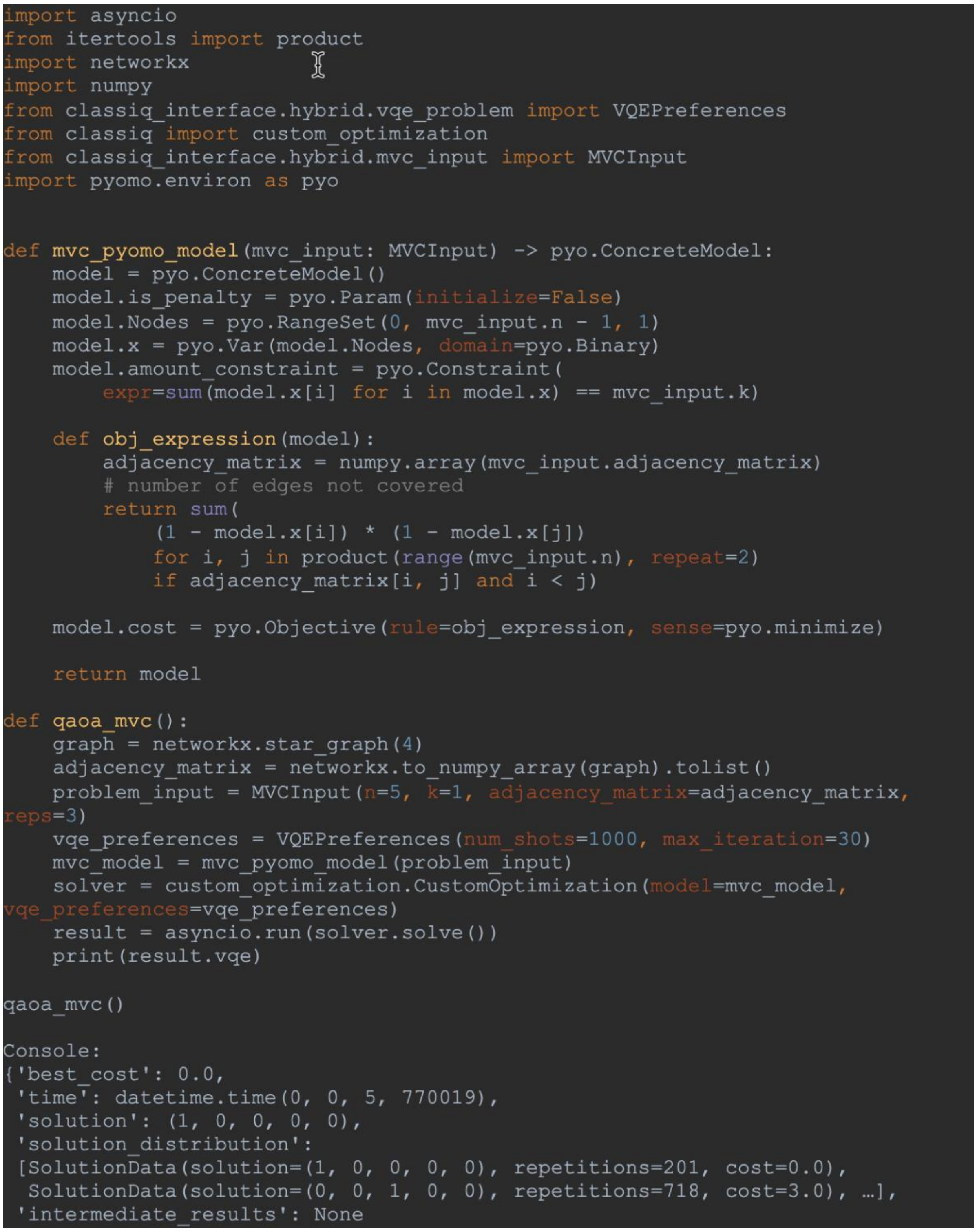
chvatal_graph() Again, note that turning any of the purple dots to yellow would violate the “no adjacency” rule for the yellow dots.	
heawood_graph()	

Here is the resultant quantum circuit for the 5-node star, presented after simplification using the Classiq ‘analyzer’ tool. The circuit uses six qubits because the graph has six vertices. Note: text annotations were added manually for the purpose of this application note:



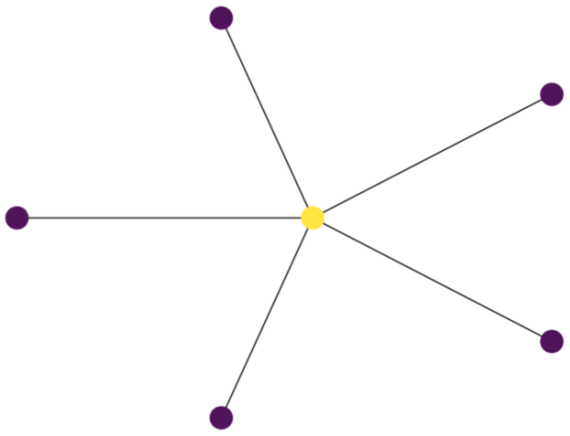
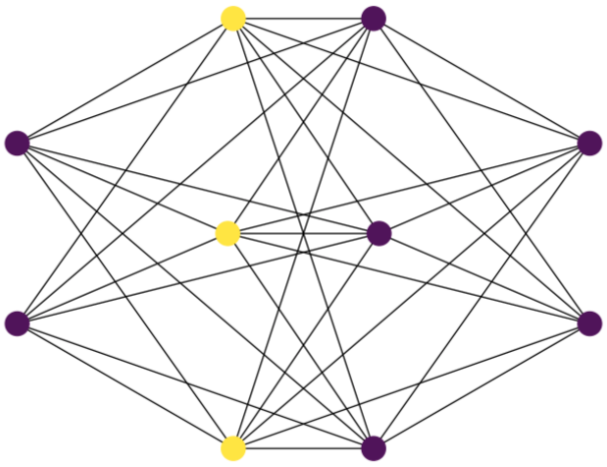
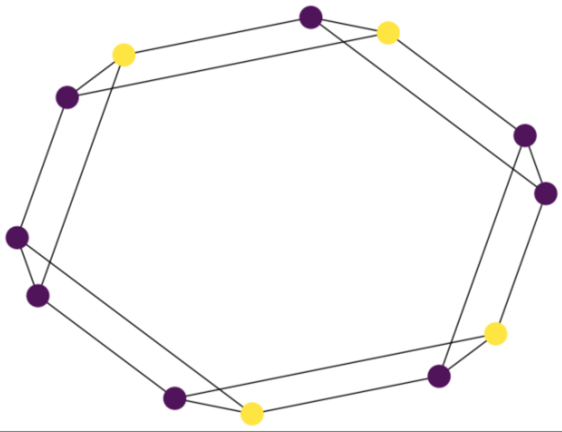
Graph Optimization: Max Vertex Cover

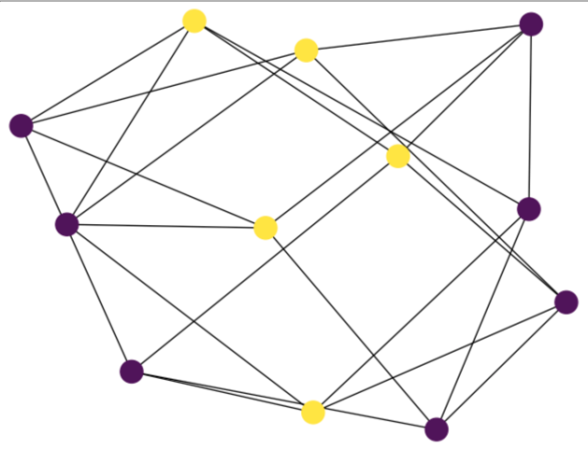
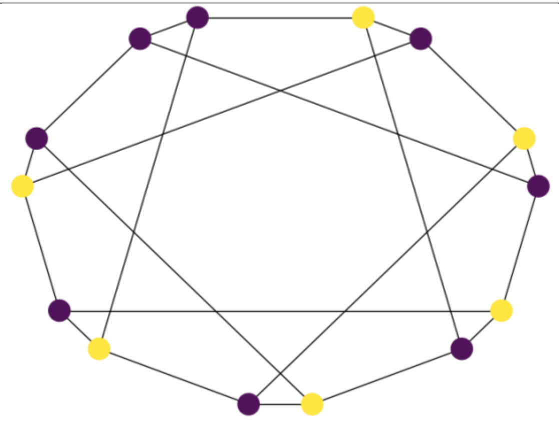
The Max Vertex Cover is a similar problem (refer to the previous section). Given a graph (e.g., the airport structure) and a constant number K (e.g., the number of available cameras) the goal is to find the best K vertices that give the maximum coverage for the graph. An edge (e.g., corridor) would be called 'covered' if at least of its vertices is selected (e.g., has a camera). The setup is similar, except for the cost function:



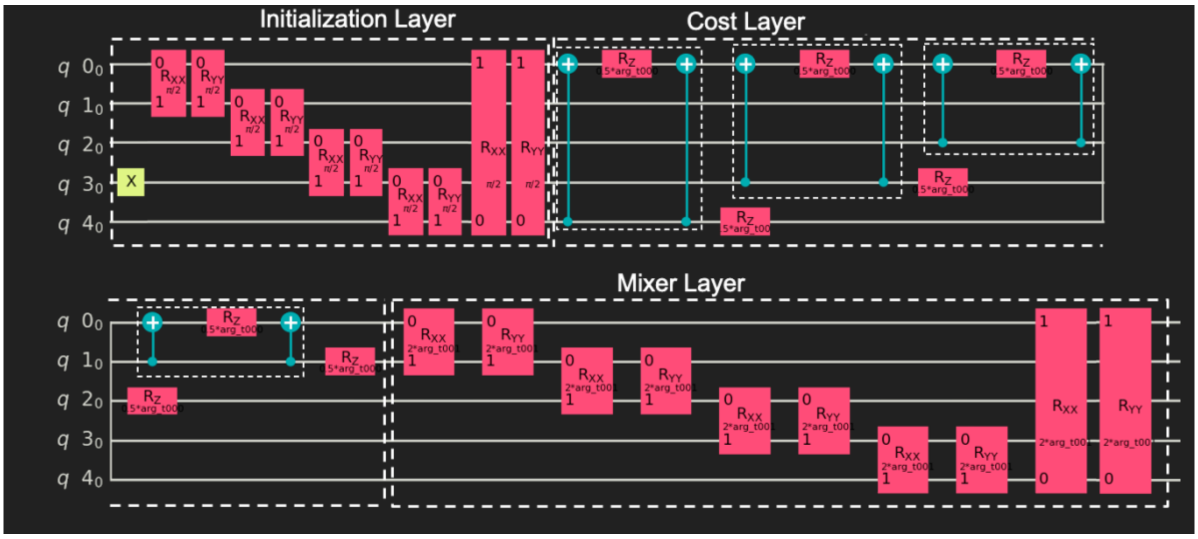
```
import asyncio
from itertools import product
import networkx
import numpy
from classiq_interface.hybrid.vqe_problem import VQEPreferences
from classiq import custom_optimization
from classiq_interface.hybrid.mvc_input import MVCInput
import pyomo.environ as pyo


def mvc_pyomo_model(mvc_input: MVCInput) -> pyo.ConcreteModel:
    model = pyo.ConcreteModel()
    model.is_penalty = pyo.Param(initialize=False)
    model.Nodes = pyo.RangeSet(0, mvc_input.n - 1, 1)
    model.x = pyo.Var(model.Nodes, domain=pyo.Binary)
    model.amount_constraint = pyo.Constraint(
        expr=sum(model.x[i] for i in model.x) == mvc_input.k)

    def obj_expression(model):
        adjacency_matrix = numpy.array(mvc_input.adjacency_matrix)
        # number of edges not covered
        return sum(
            (1 - model.x[i]) * (1 - model.x[j])
            for i, j in product(range(mvc_input.n), repeat=2)
            if adjacency_matrix[i, j] and i < j)

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

    return model

def qaoa_mvc():
    graph = networkx.star_graph(4)
    adjacency_matrix = networkx.to_numpy_array(graph).tolist()
    problem_input = MVCInput(n=5, k=1, adjacency_matrix=adjacency_matrix,
reps=3)
    vqe_preferences = VQEPreferences(num_shots=1000, max_iteration=30)
    mvc_model = mvc_pyomo_model(problem_input)
    solver = custom_optimization.CustomOptimization(model=mvc_model,
vqe_preferences=vqe_preferences)
    result = asyncio.run(solver.solve())
    print(result.vqe)

qaoa_mvc()

Console:
{'best_cost': 0.0,
 'time': datetime.time(0, 0, 5, 770019),
 'solution': (1, 0, 0, 0, 0),
 'solution_distribution':
 [SolutionData(solution=(1, 0, 0, 0, 0), repetitions=201, cost=0.0),
  SolutionData(solution=(0, 0, 1, 0, 0), repetitions=718, cost=3.0), …],
 'intermediate_results': None
```

Here are the results. In this case, yellow indicates that a camera should be placed on that vertex:

Graph type (using networkx library)	Max Vertex Cover
star_graph(5), k=1 In this case, if we have just a single camera, placing it in the middle would provide the best coverage as all five edges would be covered.	
turan_graph(10, 3) , k=3	
LCF_graph(12, [3, -3], 5) , k=4	

chvatal_graph() , k=5	
networkx.heawood_graph(), k=5	

And here is the analyzed quantum circuit:



August 2021

Summary

Optimization is a powerful technique that helps numerous markets. The Classiq platform helps companies harness the power of quantum computing to solve difficult optimization problems.

CLASSIQ

REVOLUTIONIZING THE DEVELOPMENT OF QUANTUM SOFTWARE

In this note, you learned about optimization using quantum circuits and using the Classiq platform. We hope you had a chance to appreciate how much a QAD platform makes it easier to design sophisticated quantum algorithms.

REQUEST A DEMO TODAY

hello@classiq.io
www.classiq.io