ESTUDO DO PROBLEMA DO CAIXEIRO VIAJANTE UTILIZANDO COMPUTAÇÃO QUÂNTICA

Luíza Liao (IC) e Calebe de Paula Bianchini (Orientador).

Apoio:PIVIC Mackenzie

RESUMO

Este trabalho apresenta um estudo do Problema do Caixeiro Viajante (*Traveling Salesman Problem* – TSP) utilizando computação quântica. O TSP é um problema clássico de otimização combinatória cujo objetivo é encontrar uma rota de menor custo que visite todas as cidades uma única vez e retorne à cidade de origem. Por ser um problema NP-difícil, sua resolução exata torna-se computacionalmente custosa à medida que aumenta o número de cidades. Neste estudo, comparamos dois algoritmos híbridos, compostos por componentes clássicos e quânticos: o *Quantum Approximate Optimization Algorithm* (QAOA) e o *Variational Quantum Eigensolver* (VQE). Foram consideradas instâncias com três e quatro cidades, executadas tanto em um simulador ideal do Qiskit quanto em hardware quântico da IBM. A comparação considera o tempo de execução e o custo das soluções obtidas. Para três cidades, ambos os algoritmos encontraram soluções de mesmo custo nos dois ambientes. Para quatro cidades, o VQE manteve o mesmo custo no simulador e no hardware real, enquanto o QAOA apresentou uma solução de maior custo no hardware real. Nos experimentos realizados, o VQE também apresentou menores tempos de execução que o QAOA.

Palavras-chave: Problema do Caixeiro Viajante. Computação quântica. Otimização combinatória.

ABSTRACT

This work presents a study of the Traveling Salesman Problem (TSP) using quantum computing. The TSP is a classical combinatorial optimization problem whose objective is to find a minimum-cost route that visits each city exactly once and returns to the starting city. Since the TSP is NP-hard, finding exact solutions becomes computationally expensive as the number of cities increases. In this study, we compare two hybrid quantum-classical algorithms: the Quantum Approximate Optimization Algorithm (QAOA) and the Variational Quantum Eigensolver (VQE). Instances with three and four cities were evaluated using both an ideal Qiskit simulator and IBM quantum hardware. The algorithms were compared in terms of execution time and the cost of the solutions obtained. For three cities, both algorithms obtained solutions with the same cost in both execution environments. For four cities, VQE maintained the same solution cost in the simulator and on real quantum hardware, whereas QAOA

produced a higher-cost solution on the quantum hardware. In the experiments performed, VQE also exhibited shorter execution times than QAOA.

Keywords: Traveling Salesman Problem. Quantum computing. Combinatorial Optimization.

1. INTRODUÇÃO

O Problema do Caixeiro Viajante, também conhecido como *Traveling Salesman Problem* (TSP) é um problema clássico de otimização combinatória. O TSP consiste em indicar a um caixeiro viajante o caminho mais curto possível para que ele passe por n cidades. Apesar disso, o problema restringe essa rota, pois ele pode passar apenas uma vez por cada cidade, além de retornar, ao final, à cidade de origem. É utilizado uma modelagem baseada em grafos para representar o mapa, sendo que os vértices representam as cidades que o caixeiro precisa visitar e as arestas simulam as rotas que serão seguidas (Talbi; Draa; Batouche, 2004).

Os matemáticos William Rowan Hamilton e Thomas Penyngton Kirkman foram os primeiros a estudar conceitos para o TSP, no século XIX. Desde então, o estudo do tema ainda é usado pela comunidade acadêmica, como base para pesquisa de fórmulas e modos para resolver problemas gerais de otimização, e, por isso, a sua relevância. O TSP possui diversas aplicabilidades em cenários reais, como por exemplo em coleta de lixo urbano, em rota de veículos, entregas de encomendas, dentre outros (Klug, 2024).

A parte intrigante desse problema é a sua complexidade computacional. Este problema é classificado como NP-Difícil, o que significa que é um problema complexo e ainda não há conhecimento de como resolver em um tempo polinomial. Existem algoritmos clássicos que solucionam o Problema do Caixeiro Viajante, mas o tempo de sua resolução é exponencial de acordo com o número de cidades. Apesar disso, existem alguns algoritmos conhecidos por não serem tão precisos nas soluções desse problema, pois utilizam otimização, mas eles conseguem considerar uma quantidade grande de cidades no TSP (Goswami et al., 2024).

Atualmente, a computação quântica se encontra na era conhecida como NISQ (*Noisy Intermediate-Scale Quantum*), onde os computadores quânticos ainda são suscetíveis a ruídos e possuem um número limitado de qubits lógicos. Porém, a era NISQ representa um passo importante na evolução da computação quântica, antes de se atingir a escala e a precisão necessárias para a computação quântica de tolerância a erros e uso em larga escala. Por isso, a utilização de algoritmos híbridos (clássicos e quânticos) é a mais adequada neste momento, por utilizar computadores quânticos somente quando necessário, ajudando a reduzir as chances de ocorrência de falha de precisão (Qian et al., 2023).

Mesmo com a evolução dos algoritmos clássicos para a resolução do Problema do Caixeiro Viajante (TSP), ainda não é possível resolvê-lo com precisão utilizando algoritmos clássicos conforme o número de cidades vai aumentando, devido à sua complexidade computacional NP-difícil. A modelagem do TSP como a busca por ciclos hamiltonianos

mínimos em grafos completos é consolidada, mas continua apresentando grandes desafios computacionais (Goswami et al., 2024).

Aproveitando os princípios da computação quântica, como a superposição de qubits, e algoritmos como o *Quantum Approximate Optimization Algorithm* (QAOA) e o *Variational Quantum Eigensolver* (VQE), é relevante investigar até onde os algoritmos quânticos conseguem encontrar soluções próximas ao ótimo em instâncias do TSP modeladas como grafos (Klug, 2024). Assim, o problema de pesquisa é comparar os resultados do TSP nos algoritmos híbridos QAOA e VQE.

O estudo do Problema do Caixeiro Viajante representa o estudo de problemas de otimização combinatória, onde há inúmeros casos de características semelhantes que podem ser mais bem resolvidos de forma parecida, além de ser um problema de logística muito lucrativo considerando a rentabilidade do mercado de logística e otimização. Mesmo sabendo que os algoritmos clássicos e aproximados já cumprem bem o seu papel em criar rotas que passem em todos os pontos necessários, ainda possuem suas limitações: a quantidade de características para uma tomada de decisão (Pop et al., 2024).

Inspirados nessas limitações e na complexidade computacional, pesquisadores e desenvolvedores estudam meios de solucionar esse problema. Com o avanço da área da computação quântica, surgiu a hipótese de recorrer a computadores quânticos para possivelmente obter melhores resultados considerando seus recursos, mesmo que ainda seja uma área em crescimento e em estágios iniciais e ainda mais teórica do que prática (Preskill, 2018).

Portanto, considerando o cenário atual da área da computação quântica, o estudo do Problema do Caixeiro Viajante utilizando computação quântica é uma oportunidade de aprofundar o conhecimento em uma tecnologia emergente, que por ser uma área nova, ainda precisa capacitar muitos profissionais e captar a atenção de mais pessoas para essa nova tecnologia.

2. DESENVOLVIMENTO DO ARGUMENTO

2.1 Formulação QUBO

Para reproduzir o QAOA e o VQE é necessário estruturá-los para posteriormente otimizá-los. A formulação do TSP é a mesma para os dois algoritmos que serão apresentados, inicia-se a partir do *Quadratic Unconstrained Binary Optimization* (QUBO). O QUBO é a formulação matemática que descreve o problema utilizando variáveis de decisão binárias, geralmente representadas por 0 e 1, para modelar o problema que deverá ser

otimizado de maneira padronizada (Glover; Kochenberger; Du, 2019). A princípio é necessário avaliar as restrições do problema, neste caso há duas restrições, cada cidade deve aparecer em apenas uma posição da formulação e cada posição deve conter apenas uma cidade. Posteriormente, deve-se adicionar uma penalidade para as situações que não cumprem as restrições, a fim de garantir que as soluções satisfaçam os critérios. As distâncias entre as cidades são apresentadas como custo, onde o objetivo é obter o menor custo. Assim, o problema deixa de ser tratado apenas como uma enumeração de caminhos e passa a ser formulado como a minimização de uma função quadrática binária (Papalitsas et al., 2019).

Figura 1. Implementação da penalidade de acordo com a restrição de posição das cidades.

```
#PENALIDADE Hposicao

for p in range(numero_cidades):

    # termos lineares
    for i in range(numero_cidades):
        k = indices[(i, p)]
        H_posicao[k][k] += -A

    # termos quadráticos
    for i in range(numero_cidades):
        for j in range(i + 1, numero_cidades):

            k1 = indices[(i, p)]
            k2 = indices[(j, p)]

            H_posicao[k1][k2] += A
            H_posicao[k2][k1] += A
```

Fonte:. Elaborado pelo autor.

Figura 2. Implementação da penalidade de acordo com a restrição que impede a repetição de cidades.

```
# PENALIDADE Hcidade

for i in range(numero_cidades):

    # termos lineares
    for p in range(numero_cidades):
        k = indices[(i, p)]
        H_cidade[k][k] += -A

    # termos quadráticos
    for p in range(numero_cidades):
        for q in range(p + 1, numero_cidades):

            k1 = indices[(i, p)]
            k2 = indices[(i, q)]

            H_cidade[k1][k2] += A
            H_cidade[k2][k1] += A
```

Fonte:. Elaborado pelo autor.

Figura 3. Implementação do peso das penalidades**.**

```
#PESO DA PENALIDADE

cada_linha = []

for l in matriz_distancias:
    maior = max(l)
    cada_linha.append(maior)

maior_distancia = max(cada_linha)

A = maior_distancia * 3
```

Fonte:. Elaborado pelo autor.

Figura 4. Implementação do custo da distância entre as cidades.

```
# CUSTO Hdistancia

for p in range(numero_cidades):

    proxima_posicao = (p + 1) % numero_cidades

    for i in range(numero_cidades):
        for j in range(numero_cidades):
            k1 = indices[(i, p)]
            k2 = indices[(j, proxima_posicao)]

            distancia = matriz_distancias[i][j]

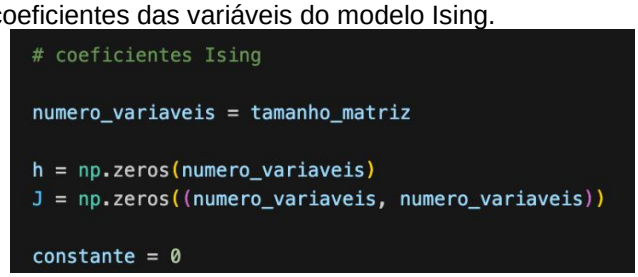
            H_distancia[k1][k2] += distancia / 2
            H_distancia[k2][k1] += distancia / 2
```

Fonte:. Elaborado pelo autor.

2.2 Modelo Ising

A transformação linear do QUBO em modelo Ising é realizada a fim de tornar a representação clássica do problema em uma maneira que seja compatível com a computação quântica. As variáveis de decisão binárias do QUBO são convertidas em variáveis de spin, representadas por -1 e +1. Os valores da conversão são associados com os autovalores de operadores de Pauli-Z, a fim de estruturar o Hamiltoniano do problema (Hp). O Hamiltoniano é o operador que servirá como referência para o ajuste de parâmetros que ocorre no otimizador clássico (Symons et al., 2023).

Figura 5. Estrutura de coeficientes das variáveis do modelo Ising.



```
# coeficientes Ising

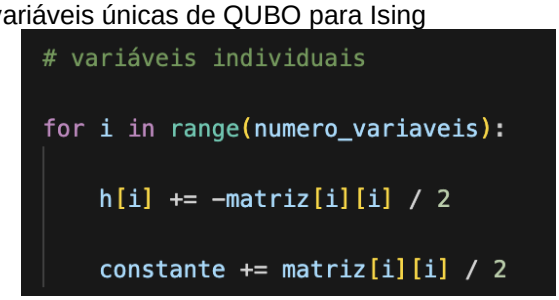
numero_variaveis = tamanho_matriz

h = np.zeros(numero_variaveis)
J = np.zeros((numero_variaveis, numero_variaveis))

constante = 0
```

Fonte:. Elaborado pelo autor.

Figura 6. Conversão das variáveis únicas de QUBO para Ising



```
# variáveis individuais

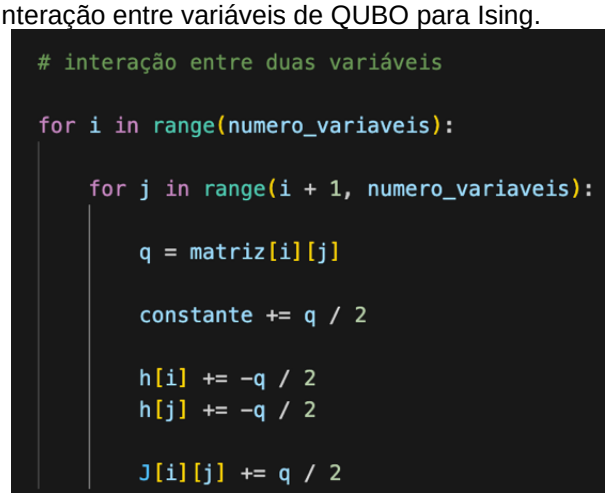
for i in range(numero_variaveis):

    h[i] += -matriz[i][i] / 2

    constante += matriz[i][i] / 2
```

Fonte:. Elaborado pelo autor.

Figura 7. Conversão de interação entre variáveis de QUBO para Ising.



```
# interação entre duas variáveis

for i in range(numero_variaveis):

    for j in range(i + 1, numero_variaveis):

        q = matriz[i][j]

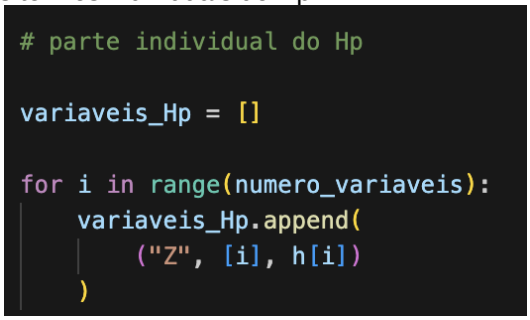
        constante += q / 2

        h[i] += -q / 2
        h[j] += -q / 2

        J[i][j] += q / 2
```

Fonte:. Elaborado pelo autor.

Figura 8. Implementação dos termos individuais do Hp.



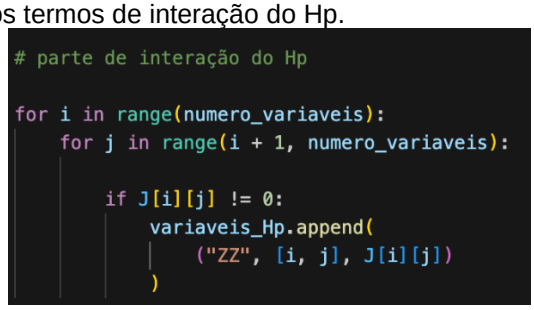
```
# parte individual do Hp

variaveis_Hp = []

for i in range(numero_variaveis):
    variaveis_Hp.append(
        ("Z", [i], h[i])
    )
```

Fonte:. Elaborado pelo autor.

Figura 9. Implementação dos termos de interação do Hp.



```
# parte de interação do Hp

for i in range(numero_variaveis):
    for j in range(i + 1, numero_variaveis):

        if J[i][j] != 0:
            variaveis_Hp.append(
                ("ZZ", [i, j], J[i][j])
            )
```

Fonte:. Elaborado pelo autor.

Figura 10. Implementação do Hp completo.



```
#Hp completo

Hp = SparsePauliOp.from_sparse_list(
    variaveis_Hp,
    num_qubits=numero_variaveis
)
```

Fonte:. Elaborado pelo autor.

2.3 QAOA

O primeiro algoritmo explorado no estudo chama-se QAOA. Esse algoritmo possui um circuito parametrizado utilizado para gerar candidatos à solução, enquanto um otimizador clássico ajusta a cada iteração os parâmetros do circuito a partir dos valores medidos. Essa característica torna o QAOA adequado ao cenário atual de dispositivos quânticos ruidosos e de escala intermediária, também conhecidos como dispositivos NISQ, pois parte do processo de busca permanece sob responsabilidade de uma etapa clássica (Preskill 2018, Blekos et al., 2024).

A reprodução do QAOA ocorre a partir de múltiplas camadas e ajustes de parâmetros. Dois Hamiltonianos participam do processo, o Hamiltoniano do problema (Hp) que tem como objetivo codificar a função objetivo e o Hamiltoniano mixer (Hm) cujo papel é garantir que o espaço de opções das soluções seja devidamente explorado. O Hm é implementado a partir da transição de estados feito por operadores de Pauli-X (Blekos et al., 2024).

Figura 11. Implementação do Hm completo.

```
#Hm

variaveis_Hm = []

for i in range(numero_variaveis):
    variaveis_Hm.append(
        ("X", [i], 1.0)
    )

Hm = SparsePauliOp.from_sparse_list(
    variaveis_Hm,
    num_qubits=numero_variaveis
)
```

Fonte:. Elaborado pelo autor.

O circuito então é construído de maneira que os Hamiltonianos são aplicados alternadamente e os parâmetros são encontrados e ajustados por um otimizador clássico. A cada iteração feita, ambos Hamiltonianos são aplicados com novos parâmetros, com o objetivo de obter melhora na otimização feita (Ruan et al., 2020).

Figura 12. Aplicação dos Hamiltonianos no circuito do QAOA.

```
# Camadas do QAOA
for camada in range(p):

    # Aplicação do Hp

    # Termos individuais
    for i in range(numero_variaveis):
        if h[i] != 0:
            circuit.rz(
                2 * gammas[camada] * h[i],
                i
            )
    # Termos de interação
    for i in range(numero_variaveis):
        for j in range(i + 1, numero_variaveis):
            if J[i][j] != 0:
                circuit.cx(i, j)
                circuit.rz(
                    2 * gammas[camada] * J[i][j],
                    j
                )
                circuit.cx(i, j)

    # Aplicação do Hm
    for i in range(numero_variaveis):
        circuit.rx(
            2 * betas[camada],
            i
        )
    circuit.barrier()
```

Fonte:. Elaborado pelo autor.

Figura 13. Ajuste dos parâmetros QAOA.

```python
# Ajuste dos parâmetros
def funcao_custo(parametros):

    valores_gamma = parametros[:p]
    valores_beta = parametros[p:]

    valores = {}

    for camada in range(p):
        valores[gammas[camada]] = valores_gamma[camada]
        valores[betas[camada]] = valores_beta[camada]

    circuito_parametrizado = circuit.assign_parameters(valores)

    estado = Statevector.from_instruction(circuito_parametrizado)

    energia = estado.expectation_value(Hp)

    return energia.real
```

Fonte:. Elaborado pelo autor.

Figura 14. Resultado da otimização do QAOA.

```python
#Resultado otimização
inicio_qaoa = time.perf_counter()

resultado = minimize(
    funcao_custo,
    parametros_iniciais,
    method="COBYLA"
)

fim_qaoa = time.perf_counter()

tempo_qaoa = fim_qaoa - inicio_qaoa
```

Fonte:. Elaborado pelo autor.

2.4 VQE

O outro algoritmo híbrido estudado foi o VQE, ele utiliza quatro componentes para otimizar o problema abordado, os componentes são: o Hamiltoniano responsável por descrever o problema que desejamos otimizar, o ansatz que é quem prepara o estado quântico no circuito, o estimador calcula o valor esperado do Hamiltoniano do problema e o otimizador clássico ajusta os parâmetros do circuito para a busca de melhores resultados. O VQE também se apresenta como um potencial candidato para uso na era NISQ (Fedorov et al., 2021).

O início da estrutura do VQE ocorre na construção do ansatz, função de onda com parâmetros ajustáveis. O ansatz é responsável por preparar o estado quântico no circuito do VQE. Os parâmetros são ajustados a cada iteração em busca de minimizar o valor esperado do Hamiltoniano (Fedorov et al., 2021). O número de parâmetros foi determinado pelo número de qubits e camadas de rotação, baseadas no método hardware-efficient, que utiliza rotações

parametrizadas intercaladas com operações de emaranhamento entre os qubits (Kandala et al., 2017).

Figura 15. Repetições do ansatz.

```
# Número de repetições do ansatz
reps = 5

# Número de parâmetros
numero_parametros = numero_variaveis * (reps + 1)

thetas = ParameterVector("theta", numero_parametros)
```

Fonte:. Elaborado pelo autor.

Figura 16. Construção do ansatz no VQE.

```
# Circuito do VQE
circuit_vqe = QuantumCircuit(numero_variaveis)

indice_theta = 0

# Camada de rotações
for i in range(numero_variaveis):
    circuit_vqe.ry(thetas[indice_theta], i)
    indice_theta += 1

# Repetição de camadas
for repeticao in range(reps):

    # Emaranhamento
    for i in range(numero_variaveis - 1):
        circuit_vqe.cx(i, i + 1)

    # Nova camada de rotações
    for i in range(numero_variaveis):
        circuit_vqe.ry(thetas[indice_theta], i)
        indice_theta += 1
```

Fonte:. Elaborado pelo autor.

Figura 17. Avaliação dos parâmetros do ansatz a partir do valor esperado de Hp no VQE.

```
# Aplicação de parâmetros do ansatz
def funcao_custo_vqe(parametros):

    valores = {
        thetas[i]: parametros[i]
        for i in range(numero_parametros)
    }

    circuito_parametrizado = circuit_vqe.assign_parameters(valores)

    estado = Statevector.from_instruction(circuito_parametrizado)

    energia = estado.expectation_value(Hp)

    return energia.real
```

Fonte:. Elaborado pelo autor.

Figura 18. Otimização dos parâmetros do ansatz no VQE.

```
# Otimização dos parâmetros
inicio_vqe = time.perf_counter()

resultado_vqe = minimize(
    funcao_custo_vqe,
    parametros_iniciais_vqe,
    method="COBYLA"
)

fim_vqe = time.perf_counter()

tempo_vqe = fim_vqe - inicio_vqe
```

Fonte:. Elaborado pelo autor.

2.5 Resultados dos experimentos

O experimento foi realizado com 3 e 4 cidades, sendo executado em um simulador ideal quântico fornecido pelo Qiskit e no hardware quântico IBM Marrakesh fornecido pela IBM.

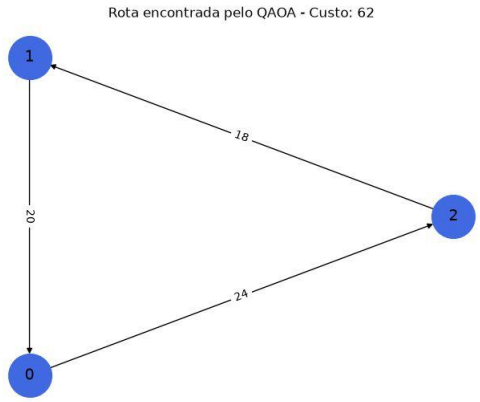
Os resultados dos experimentos com 3 cidades estão descritos na Tabela 1.

Tabela 1. Tempo de execução, custo encontrado e quantidade de iterações utilizadas para cada algoritmo com 3 cidades.

Algoritmo	Tempo (s)	Custo Total	Nº de iterações
QAOA simulado	7.16	62	10
QAOA hardware real	72	62	10
VQE simulado	2.87	62	3
VQE hardware real	6	62	3

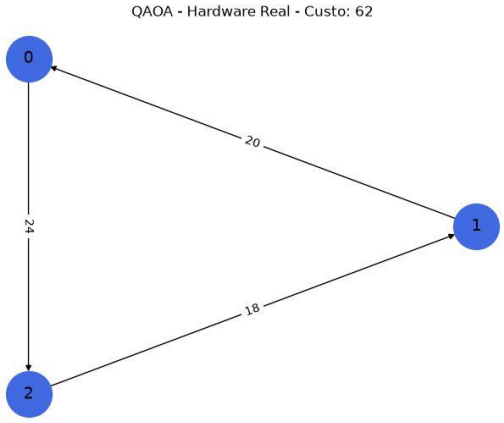
Fonte:. Elaborado pelo autor.

Figura 19. Rota obtida pelo QAOA em simulador para 3 cidades.



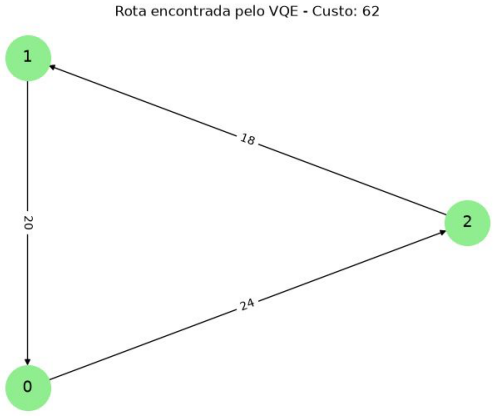
Fonte:. Elaborado pelo autor.

Figura 20. Rota obtida pelo QAOA em hardware real para 3 cidades.



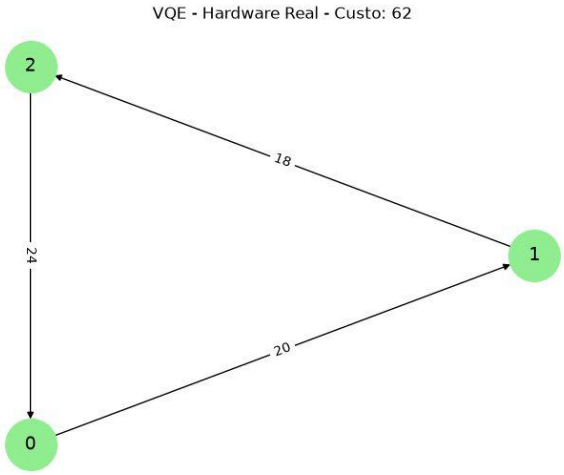
Fonte:. Elaborado pelo autor.

Figura 21. Rota obtida pelo VQE em simulador para 3 cidades.



Fonte:. Elaborado pelo autor.

Figura 22. Rota obtida pelo VQE em hardware quântico para 3 cidades.



Fonte:. Elaborado pelo autor.

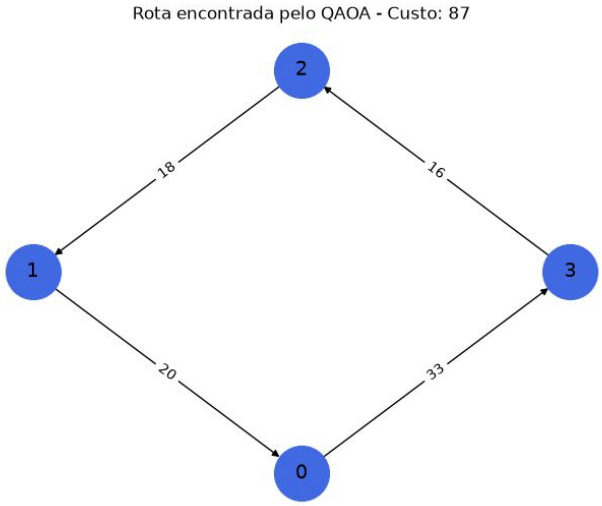
Os resultados dos experimentos com 4 cidades estão descritos na Tabela 2.

Tabela 2. Tempo de execução, custo encontrado e quantidade de iterações utilizadas para cada algoritmo com 4 cidades.

Algoritmo	Tempo (s)	Custo Total	Nº de iterações
QAOA simulado	245.9	87	10
QAOA hardware real	72	109	10
VQE simulado	44.4	87	3
VQE hardware real	2	87	3

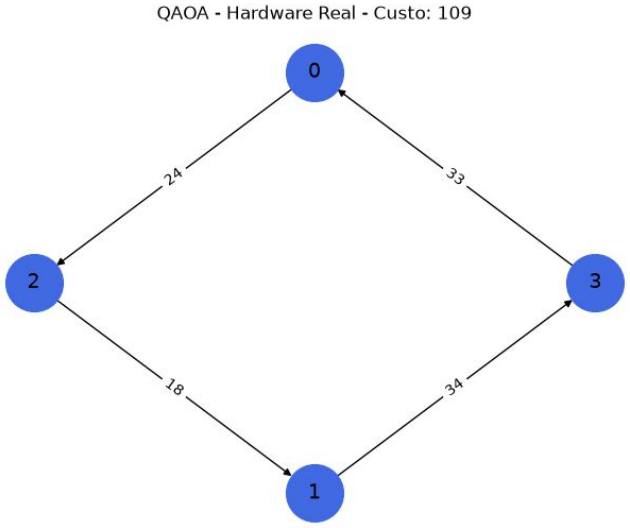
Fonte:. Elaborado pelo autor.

Figura 23. Rota obtida pelo QAOA em simulador para 4 cidades.



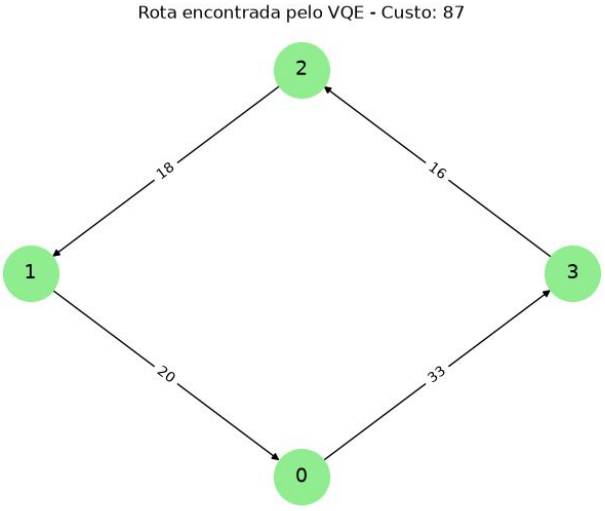
Fonte:. Elaborado pelo autor.

Figura 24. Rota obtida pelo QAOA em hardware quântico para 4 cidades.



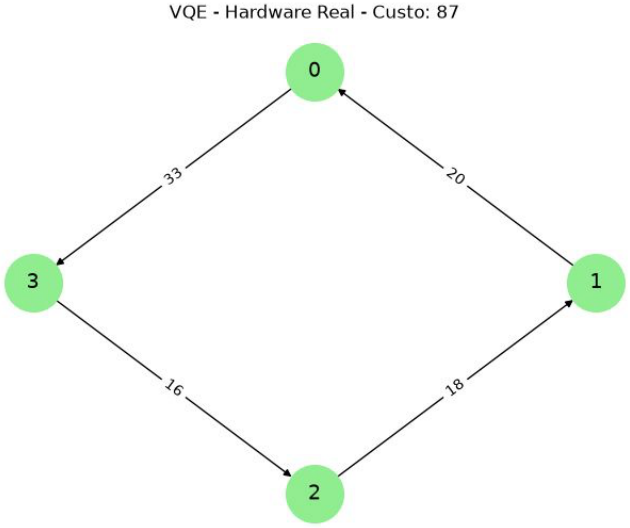
Fonte:. Elaborado pelo autor.

Figura 25. Rota obtida pelo VQE em simulador para 4 cidades.



Fonte:. Elaborado pelo autor.

Figura 26. Rota obtida pelo VQE em hardware quântico para 4 cidades.



Fonte:. Elaborado pelo autor.

Os resultados obtidos mostram que o algoritmo QAOA apresentou uma solução não ótima quando executado em hardware real em um problema com 4 cidades.

3. CONSIDERAÇÕES FINAIS

Este artigo apresenta um estudo sobre a aplicação de computação quântica ao Problema do Caixeiro viajante com ênfase em formulações QUBO, QAOA e VQE. A análise de resultados mostra que enquanto o QAOA teve alteração no hardware real o VQE não, além do VQE obter resultados com menor tempo de processamento.

Mesmo estando na era NISQ da computação quântica, os algoritmos híbridos e as formulações QUBO constituem caminhos relevantes para investigar o comportamento de técnicas quânticas em problemas de otimização combinatória. Como próximos passos, pretende-se expandir a análise para uma maior variedade de algoritmos, outros valores de quantidades de cidades uma variedade maior de TSP base para que os resultados qualitativos sejam mais abrangentes.

4. REFERÊNCIAS

BLEKOS, K. et al. A review on quantum approximate optimization algorithm and its variants. Physics Reports, v. 1068, p. 1–66, 2024. Disponível em: <https://doi.org/10.1016/j.physrep.2024.03.002>.

FEDOROV, D. A. et al. VQE Method: A short Survey and Recent Developments. Materials Theory, v. 6, n. 1. 2021. Disponível em: <https://doi.org/10.48550/arXiv.2103.08505>.

GLOVER, Fred; KOCHENBERGER, Gary; DU, Yu. Quantum Bridge Analytics I: a tutorial on formulating and using QUBO models. 4OR, v.17, n. 4, p.335-371, 2019. Disponível em: <https://doi.org/10.1007/s10288-019-00424-y>.

GOSWAMI, K. et al. Solving the travelling salesman problem using a single qubit. arXiv, 2024. Disponível em: <https://doi.org/10.48550/arXiv.2407.17207>.

KANDALA, A. et al. Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets. Nature, v. 549, n. 7671, p. 242–246, 2017. Disponível em: <https://doi.org/10.1038/nature23879>.

KLUG, F. Quantum Algorithms for Solving the Traveling Salesman Problem. SSRN, 2024. Disponível em: <http://dx.doi.org/10.2139/ssrn.4836033>.

PAPALITSAS, C. et al. A QUBO model for the traveling salesman problem with time windows. Algorithms, v. 12, n. 11, p. 224, 2019. Disponível em: <https://doi.org/10.3390/a12110224>.

PRESKILL, J. Quantum computing in the NISQ era and beyond. Quantum, v. 2, n. 2, p. 79, 2018. Disponível em: < https://doi.org/10.48550/arXiv.1801.00862>.

POP, Petrica C et al. A comprehensive survey on the generalized traveling salesman problem, European Journal of Operational Research, v. 314, n. 3, 2024. Disponível em: <https://doi.org/10.1016/j.ejor.2023.07.022>.

QIAN, W. et al. Comparative study of variations in quantum approximate optimization algorithms for the traveling salesman problem. Entropy, v. 25, n. 8, p. 1238-1238, 2023. Disponível em: <https://doi.org/10.3390/e25081238>.

RUAN, Y. et al. The quantum approximate algorithm for solving traveling salesman problem. Computers, Materials & Continua, v. 63, n. 3, p. 1237–1247, 2020. Disponível em: <https://doi.org/10.32604/cmc.2020.010001>.

SYMONS B. C. B. et al. A practitioner's guide to algorithms for optimization problems. 2023. Diponível em: <https://doi.org/10.48550/arXiv.2305.07323>.

TALBI, H.; DRAA, A.; BATOUCHE, M. A new quantum-inspired genetic algorithm for solving the travelling salesman problem. International Conference on Industrial Technology, v. 3, p. 1192–1197, 2004. Disponível em: <https://doi.org/10.1109/ICIT.2004.1490730>.

Contatos: luizaliao2222@gmail.com e calebe.bianchini@mackenzie.br