Traveling Salesman Problem: Implementation of an Optimization Algorithm based on Operational Research Techniques for Complexity Analysis

CARLOS FERNANDO TEIXEIRA LEITE, MICAEL ANDRÉ CUNHA DIAS, SÉRGIO LUÍS LOPES FÉLIX

8200377@estg.ipp.pt, 8200393@estg.ipp.pt, 8200615@estg.ipp.pt

Instituto Politécnico do Porto, Escola Superior de Tecnologia e Gestão, Rua do Curral, Casa do Curral–Margaride, 4610-156 Felgueiras, Portugal.

Abstract: Este estudo apresenta uma análise do problema do Caixeiro Viajante (TSP) e do algoritmo Nearest Neighbor (NN). O TSP é um desafio clássico de otimização, onde o objetivo é encontrar a rota mais curta que visita todas as cidades uma vez e retorna à cidade inicial.

Keywords: Traveling Salesman Problem (TSP), NP-hard (Non-deterministic Polynomial-time hard), Nearest Neighbor Algorithm (NNA)

Source Code: https://github.com/WallQ/AAO

1 Introduction

O presente trabalho prático tem como objetivo a resolução do desafiante problema do Traveling Salesman Problem (TSP), no âmbito da disciplina de Análise Algorítmica e Optimização (AAO). Neste trabalho, foi proposta a implementação de um algoritmo para solucionar o problema do caixeiro viajante, em busca de encontrar a solução ótima para cada instância de rede apresentada.

O TSP é conhecido por ser um problema NP-hard (Non-deterministic Polynomial-time hard) ou seja, encontrar uma solução ótima é computacionalmente exigente e requer tempo exponencial. Não existe um algoritmo conhecido que possa resolvê-lo para todas as entradas possíveis em tempo polinomial. À medida que o número de cidades aumenta, o número de passeios possíveis cresce exponencialmente, tornando uma busca exaustiva pela solução ótima computacionalmente inviável para instâncias de problema grandes.

Este problema tem sido estudado há décadas e várias soluções têm sido teorizadas. A abordagem mais direta é tentar todas as possibilidades, mas esse é também o método mais demorado e dispendioso. Muitas soluções utilizam heurísticas, que fornecem resultados probabilísticos. No entanto, os resultados são aproximados e nem sempre ótimos.

Uma forma de abordar esse problema é usar Algoritmos de Aproximação. Esses algoritmos fornecem soluções que são próximas do ótimo, mas não necessariamente ótimas. Em vez de focar em encontrar a rota mais eficaz, o TSP muitas vezes preocupa-se em encontrar a solução menos custosa.

Neste estudo, focamos na implementação e análise do algoritmo heurístico Nearest Neighbour Algorithm (NNA) como uma potencial solução para o TSP.

Algoritmo esse que foi proposto pelo matemático e cientista da computação britânico Ronald M. Bellman em 1946, embora a ideia de utilizar vizinhos mais próximos para resolver problemas semelhantes remonte a trabalhos anteriores.

O algoritmo NNA é relativamente simples de entender e implementar. A ideia básica é começar em uma cidade arbitrária e, em cada etapa, escolher a cidade mais próxima que ainda não foi visitada. Essa cidade é adicionada à rota e o processo é repetido até que todas as cidades tenham sido visitadas, formando assim um ciclo hamiltoniano. Por fim, a rota é fechada retornando à cidade inicial.

Embora o algoritmo NNA seja fácil de implementar, ele não garante obter a solução ótima para o problema do TSP. Ele é uma heurística gulosa, o que significa que faz escolhas ótimas em cada etapa, sem levar em consideração o impacto global dessas escolhas. Como resultado, o algoritmo pode encontrar soluções subótimas, que podem estar longe do caminho mais curto possível.

Os resultados obtidos pelo algoritmo NNA variam dependendo do número de cidades na instância do TSP. Em algumas instâncias, o algoritmo pode encontrar soluções muito próximas da solução ótima, enquanto em outras instâncias, pode encontrar soluções bastante piores, ou seja, no pior caso pode obter uma ordem de complexidade $O(N^2)$. Em geral, o algoritmo Nearest Neighbour é rápido e fornece soluções razoáveis, mas seu desempenho é superado por outros algoritmos mais avançados, como por exemplo algoritmos baseados em programação linear.

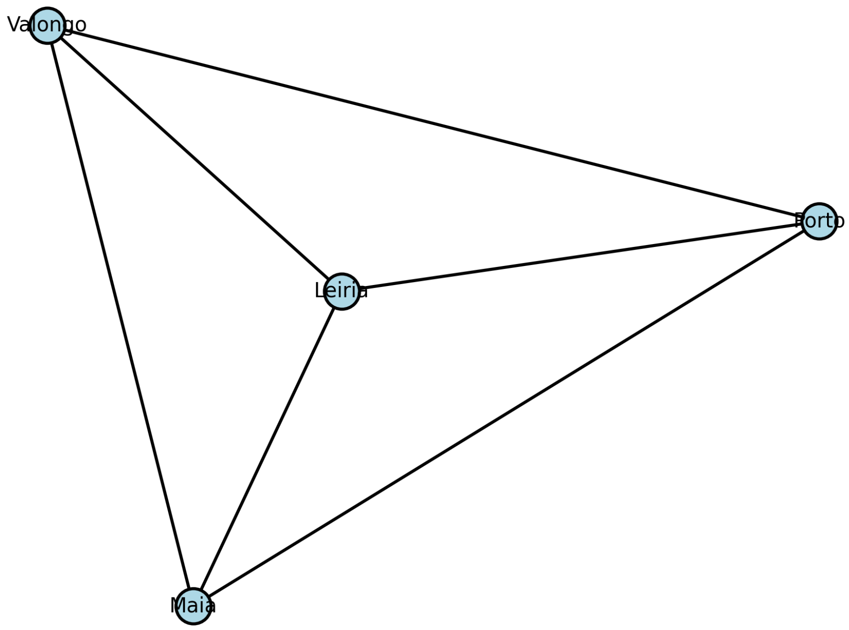
2 Materials and methods

No decorrer do desenvolvimento deste projeto criou-se uma bateria de testes com 32 cidades de Portugal, na qual utilizou-se um script para a criação das redes de forma aleatória, calculando assim a distância entre elas através da fórmula da distância euclidiana, obtendo assim as redes de forma exponencialmente, ou seja, 4 nós, 8 nós, 16 nós, 32 nós.

Após a criação das redes foi efetuado uma validação da solução ótima com base no algoritmo escolhido para a solução do problema. Ao comparar as rotas encontradas pelo NN com as rotas ótimas, é possível avaliar a qualidade da solução obtida. Pode-se calcular a diferença entre os custos das rotas encontradas e as rotas ótimas conhecidas, bem como analisar visualmente as rotas geradas para identificar eventuais desvios significativos.

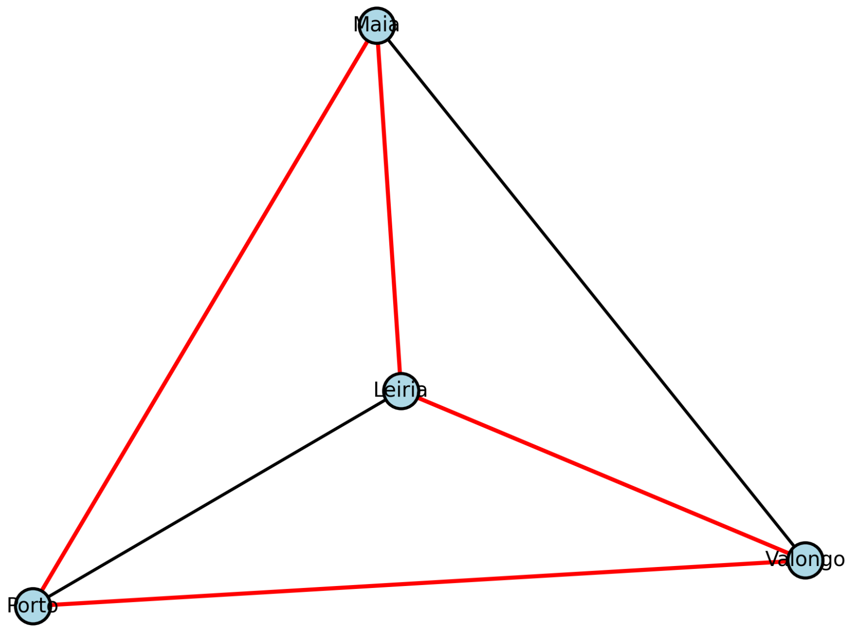
As imagens e tabelas a seguir ilustram as redes geradas e seus correspondentes caminhos ótimos:

2.1 Network com 4 nós

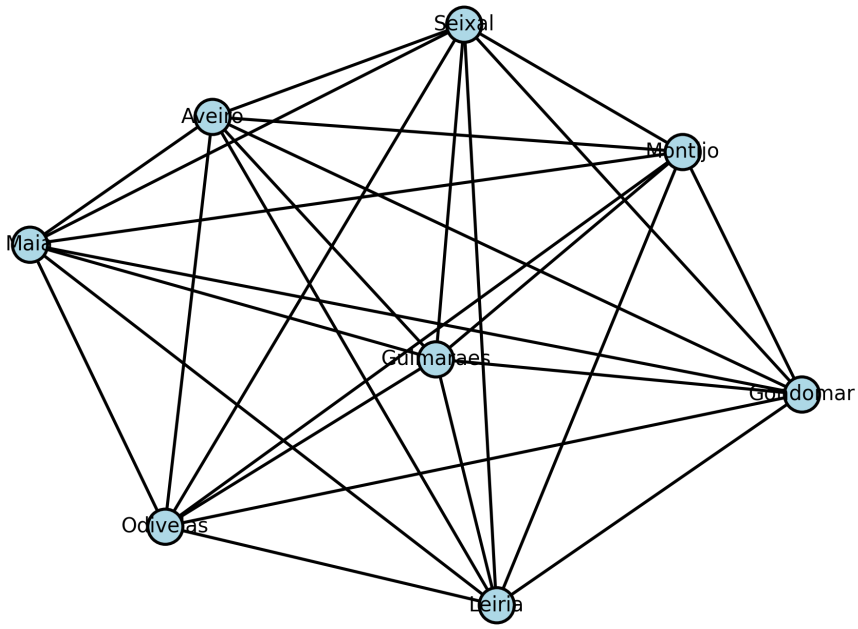


	Valongo	Porto	Leiria	Maia
Valongo	-	50	815	55
Porto	50	-	785	50
Leiria	815	785	-	830
Maia	55	50	830	-

Temos um percurso que inicia em Valongo, passa por Porto, Maia e Leiria, e retorna a Valongo. o custo de percorrer de Valongo a Porto é de 50. Em seguida, o custo de ir de Porto a Maia também é de 50. Após isso, o custo de viajar de Maia a Leiria é de 830. Por fim, o custo de retorno de Leiria a Valongo é de 815. Somando todos esses custos, temos:50 + 50 + 830 + 815 = 1745

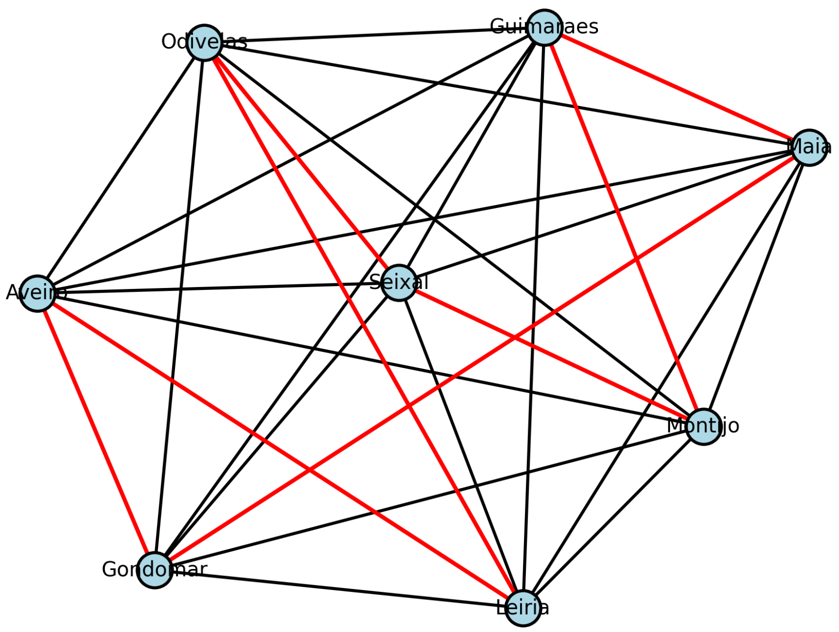


Network com 8 nós

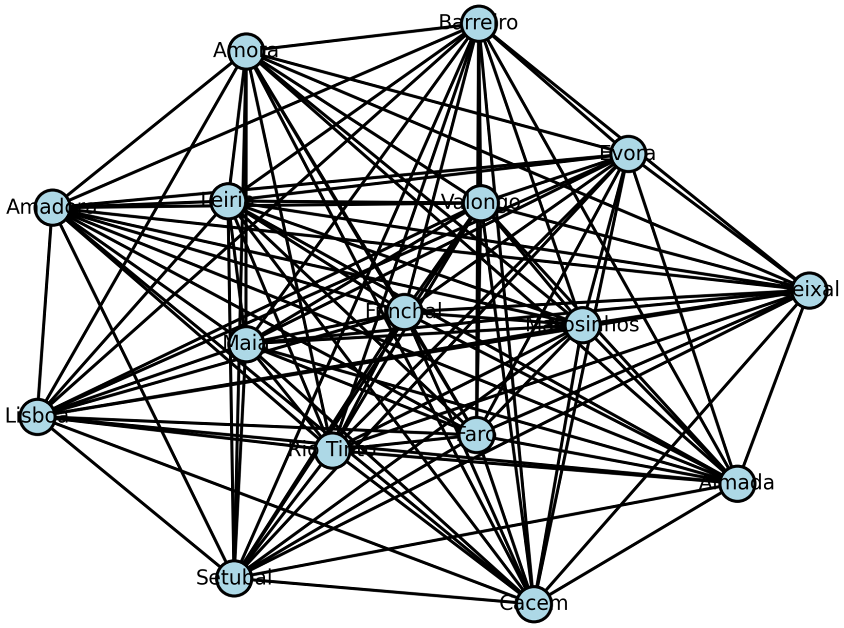


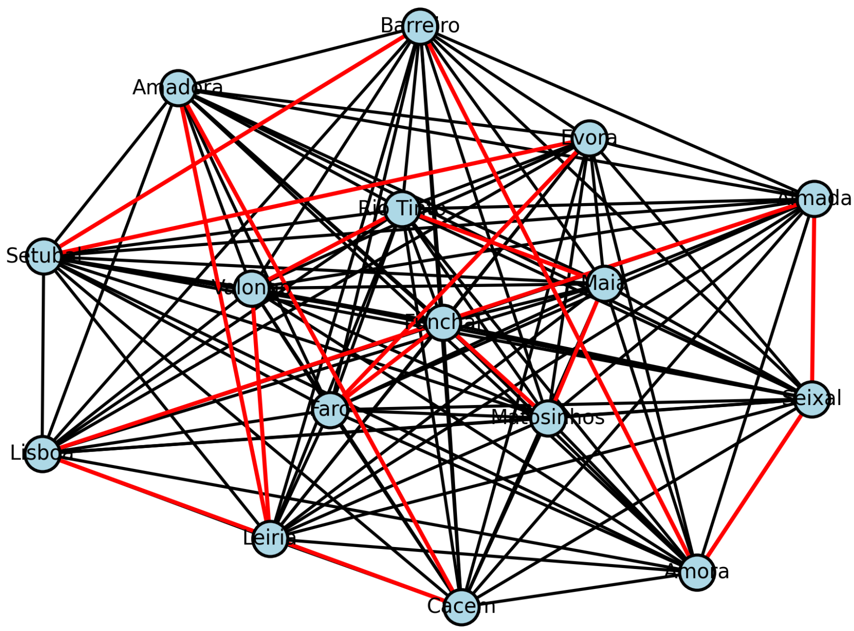
	Montijo	Odivelas	Guimarães	Gondomar	Seixal	Leiria	Aveiro	Maia
Montijo	-	105	1550	1365	65	585	1085	1415
Odivelas	105	-	1520	1335	90	555	1050	1380
Guimaraes	1550	1520	-	195	1595	965	470	180
Gondomar	1365	1335	195	-	1410	785	285	65
Seixal	65	90	1595	1410	-	630	1130	1455
Leiria	585	555	965	785	630	-	500	830
Aveiro	1085	1050	470	285	1130	500	-	330
Maia	1415	1380	180	65	1455	830	330	-

Para esta network foi desenvolvido um percurso que começa no Montijo e passa pelo Seixal, Odivelas, Leiria, Aveiro, Gondomar, Maia e Guimarães, antes de retornar ao Montijo. O custo total dessa rota é de 329 unidades monetárias.



Network com 16 nós





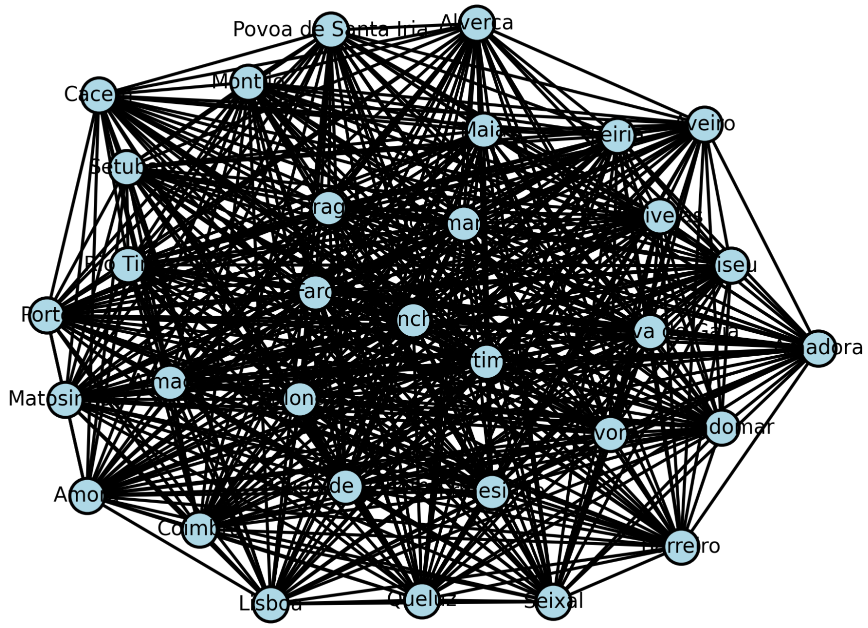
Network com 16 nós

	Matosinhos	**Leiria**	**Funchal**	**Valongo**	**Maia**	**Almada**	**Seixal**	**Évora**	**Setúbal**	**Rio Tinto**	**Cacém**	**Barreiro**	**Amadora**	**Lisboa**	**Faro**	**Amora**
Matosinhos	-	800	5980	80	40	1405	1425	1490	1480	55	1365	1410	1365	1380	2340	1440
Leiria	800	-	5360	815	830	615	630	760	680	800	585	615	580	590	1565	645
Funchal	5980	5360	-	6035	6020	4840	4845	5230	4875	6010	4825	4865	4845	4865	4760	4830
Valongo	80	815	6035	-	55	1425	1445	1480	1495	30	1390	1430	1390	1400	2335	1455
Maia	40	830	6020	55	-	1440	1455	1510	1510	40	1400	1445	1400	1415	2365	1470
Almada	1405	615	4840	1425	1440	-	30	545	145	1415	85	40	55	25	1070	40
Seixal	1425	630	4845	1445	1455	30	-	520	110	1430	115	20	90	50	1035	15
Évora	1490	760	5230	1480	1510	545	520	-	430	1475	620	510	585	540	865	525
Setúbal	1480	680	4875	1495	1510	145	110	430	-	1485	225	110	200	155	935	110
Rio Tinto	55	800	6010	30	40	1415	1430	1475	1485	-	1375	1415	1375	1390	2330	1445
Cacém	1365	585	4825	1390	1400	85	115	620	225	1375	-	120	30	80	1145	120
Barreiro	1410	615	4865	1430	1445	40	20	510	110	1415	120	-	90	45	1040	35
Amadora	1365	580	4845	1390	1400	55	90	585	200	1375	30	90	-	50	1125	95
Lisboa	1380	590	4865	1400	1415	25	50	540	155	1390	80	45	50	-	1085	60
Faro	2340	1565	4760	2335	2365	1070	1035	865	935	2330	1145	1040	1125	1085	-	1030
Amora	1440	645	4830	1455	1470	40	15	525	110	1445	120	35	95	60	1030	-

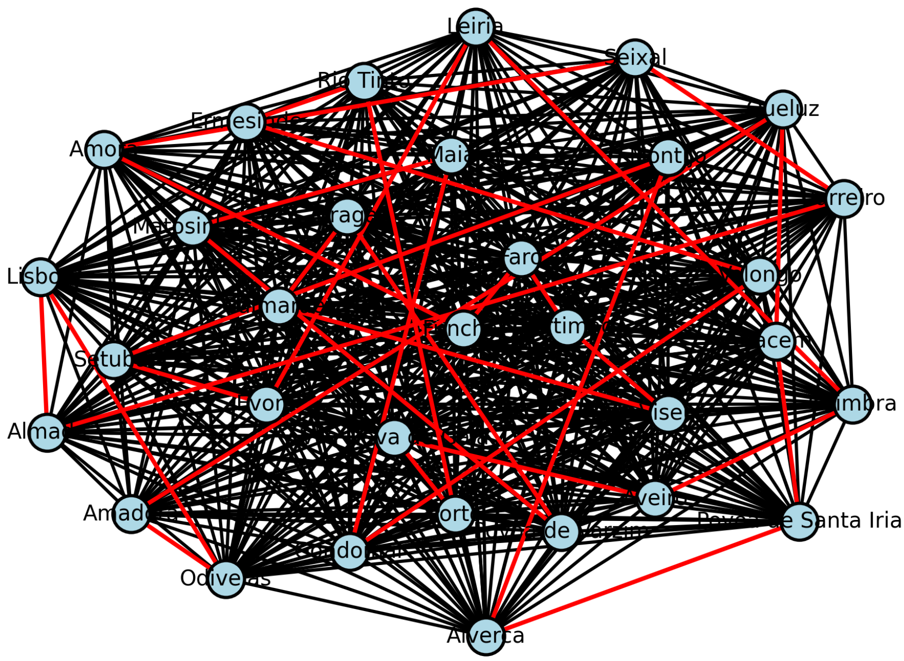
Foi desenvolvido um percurso que parte de Matosinhos e passa por Maia, Rio Tinto, Valongo, Leiria, Amadora, Cacém, Lisboa, Almada, Seixal, Amora, Barreiro, Setúbal, Évora, Faro e Funchal, antes de retornar a Matosinhos. O custo total dessa rota é de 1386 unidades monetárias, que é a soma dos custos de cada trecho ao longo do caminho.

Somando os custos temos: 40 + 40 + 30 + 815 + 580 + 30 + 80 + 25 + 30 + 15 + 35 + 110 + 430 + 865 + 4760 + 5980 = 1386

Network com 32 nós



Foi gerado um gráfico com 32 nós, embora não tenhamos criado uma tabela para acompanhar os custos entre cidades.



Implementação do Algoritmo

O código apresentado é uma implementação do algoritmo do NNA para resolver o TSP. Como entrada recebe uma rede (grafo) que representa os nós e as arestas do problema.

O código começa por selecionar o primeiro nó como o nó inicial. Em seguida, enquanto ainda existirem nós restantes para visitar, o código percorre cada um dos nós restantes e calcula a distância entre o nó atual e os nós restantes. O nó mais próximo é selecionado e adicionado ao caminho percorrido até ao momento. O custo total do caminho é atualizado com a distância mínima encontrada. Esse processo é repetido até que todos os nós tenham sido visitados.

No final, o nó inicial é adicionado novamente ao caminho, e a distância da última aresta é adicionada ao custo total. O algoritmo retorna o caminho percorrido, o custo total desse caminho e o número de comparações realizadas durante a execução do algoritmo.

Abaixo está uma imagem ilustrativa de um pequeno pedaço de código em Python que resolve o TSP:

```python
def solve_tsp(network):
    nodes = list(network.nodes)  # n
    start_node = nodes[0]  # 1
    path = [start_node]  # 1
    total_cost = 0  # 1
    remaining_nodes = nodes[1:]  # n
    comparisons = 0  # 1

    while remaining_nodes:  # n
        current_node = path[-1]  # 1
        nearest_node = None  # 1
        min_distance = float('inf')  # 1
        comparisons += 1

        for node in remaining_nodes:  # n
            distance = network[current_node][node]['weight']  # 1
            comparisons += 1
            if distance < min_distance:  # 1
                comparisons += 1
                min_distance = distance  # 1
                nearest_node = node  # 1

        path.append(nearest_node)  # n
        total_cost += min_distance  # 1
        remaining_nodes.remove(nearest_node)  # n
        comparisons += 1

    path.append(start_node)  # n
    total_cost += network[path[-2]][path[-1]]['weight']  # 1

    return path, total_cost, comparisons
```

3 Experimental Results

Na presente seção é possível visualizar a tabela com a informação referente a cada rede, cada rede possui um número de vértices e o total de comparações feita, além disso inclui o tempo de execução de cada rede como o caminho da própria e o custo associado à travessia.

É importante observar que o tempo de execução de um programa pode variar de computador para computador. O algoritmo executou numa com um processador Intel Core i5-9600K, devemos considerar que esse processador possui suas especificidades e desempenho próprio.

Vértices	Comparações	Tempo (em segundos)	Caminho	Custo
4	16	0.000036	Valongo -> Porto -> Maia -> Leiria -> Valongo	1745
8	56	0.000101	Montijo -> Seixal -> Odivelas -> Leiria -> Aveiro -> Gondomar -> Maia -> Guimarães -> Montijo	3290
16	190	0.000130	Matosinhos -> Maia -> Rio Tinto -> Valongo -> Leiria -> Amadora -> Cacém -> Lisboa -> Almada -> Seixal -> Amora -> Barreiro -> Setúbal -> Evora -> Faro -> Funchal -> Matosinhos	13865
32	638	0.000598	Amora -> Seixal -> Barreiro -> Almada -> Lisboa -> Odivelas -> Amadora -> Queluz -> Cacém -> Póvoa de Santa Iria -> Alverca -> Montijo -> Setúbal -> Évora -> Leiria -> Coimbra -> Aveiro -> Vila Nova de Gaia -> Porto -> Rio Tinto -> Ermesinde -> Valongo -> Gondomar -> Maia -> Matosinhos -> Póvoa de Varzim -> Braga -> Guimarães -> Viseu -> Portimão -> Faro -> Funchal -> Amora	15465

A complexidade de tempo da função *solve_tsp* é dominada pelo loop *while* sobre os nós restantes e pelo loop interno *for* que perforce em O(n). O número de comparações foi calculado contando as ocorrências da instruções como *for, while, if*. A complexidade de tempo permanece $O(n^2)$ para todos os tamanhos de rede, pois a complexidade algorítmica da função não muda com base no tamanho da rede. No entanto, o número de comparações aumenta quadraticamente com o tamanho da rede, seguindo a fórmula *n+1*.

Network Size (n)	Time Complexity (Big O)	Comparisons
4	O(n^2)	16
8	O(n^2)	64
16	O(n^2)	256
32	O(n^2)	1024

4 Discussion

Levando em consideração a implementação do algoritmo Nearest Neighbour Algorithm (NNA) para resolver o Problema do Caixeiro Viajante (TSP) em redes com 4, 8, 16 e 32 nós, é importante destacar algumas considerações na procura pela solução para a resolução do problema.

Este algoritmo permite encontrar uma solução num tempo relativamente curto, no entanto, por se tratar de uma heurística, não há garantia de que a solução encontrada seja a melhor possível, o que indica a possibilidade de haver margem para melhorias.

Ressalta-se que o algoritmo NNA possui uma complexidade assintótica de $O(n^2)$, o que significa que o desempenho piora significativamente à medida que o número de nós aumenta.

Em termos de resultados experimentais, os tempos de execução foram relativamente rápidos para os tamanhos de rede fornecidos, o que torna esta solução viável. No entanto, é fundamental considerar que o tempo de execução pode variar dependendo do ambiente de hardware e software.

Em geral, o trabalho apresenta uma implementação satisfatória do algoritmo NNA para solucionar o TSP em redes com 4, 8, 16 e 32 nós.

5 Conclusion

Foram realizados testes com redes de diferentes tamanhos, variando de 4 a 32 nós, para avaliar a qualidade das soluções encontradas pelo NNA. Os resultados experimentais demonstraram que o NNA foi capaz de encontrar soluções próximas ao ótimo para as diferentes instâncias de rede testadas.

É importante ressaltar que, embora o NNA seja uma solução aproximada, ele pode ser aplicado em muitos cenários práticos em que a solução ótima não é estritamente necessária. Sua simplicidade de implementação e eficiência tornam-no uma opção viável para encontrar rotas aproximadas nas nossas instâncias do TSP.

A implementação do NNA apresentado neste trabalho fornece uma abordagem razoável para resolver o TSP em redes de pequeno a médio porte. No entanto, para redes maiores, algoritmos alternativos devem ser considerados para alcançar soluções mais ótimas, como por exemplo:

- Algoritmo 2-OPT
- Algoritmo de Branch and Bound
- Algoritmo de Programação Linear

Em resumo, este trabalho contribuiu para a compreensão e aplicação do algoritmo NNA na resolução do TSP. Os resultados obtidos fornecem percepções sobre a eficácia do algoritmo para redes de diferentes tamanhos. Com a continuação da pesquisa nessa área, é possível aprimorar ainda mais as soluções aproximadas para o TSP.

References

1. *What is the traveling salesman problem (TSP)?: Definition from TechTarget, WhatIs.com.* https://www.techtarget.com/whatis/definition/traveling-salesman-problem (Accessed: 21 June 2023).

2. *Yenigün, Okan. "Traveling Salesman Problem: Nearest Neighbor Algorithm Solution",* https://blog.devgenius.io/traveling-salesman-problem-nearest-neighbor-algorithm-solution-e78399d0ab0c (Accessed 21 June 2023)