



Trabalho Prático 1

Problema do Caixeiro Viajante

Licenciatura em Engenharia Informática
UC 40437 - Algoritmos e Estruturas de Dados
Ano letivo 2018/2019

Trabalho realizado por:

88808 - João Miguel Nunes de Medeiros e Vasconcelos
88886 - Tiago Carvalho Mendes

Aveiro, 31 de dezembro de 2018

Índice

Índice

Introdução.....	3
Formulação do problema.....	4
Aplicação do problema à programação.....	7
Análise de resultados.....	8
Notas importantes e limitações.....	23
Conclusões.....	24
Referências.....	24
Anexo.....	25

Introdução

O presente documento destina-se a descrever detalhadamente a abordagem utilizada para resolver o problema proposto como primeiro trabalho prático da unidade curricular de Algoritmos e Estruturas de Dados da Universidade de Aveiro, o famoso *Problema do Caixeiro Viajante*.

Ora, o *Problema do Caixeiro Viajante* ("The Traveling Salesman Problem") é um problema que, partindo de uma cidade inicial, se pretende saber qual o melhor trajeto a percorrer visitando todas as cidades de um determinado conjunto apenas uma vez, voltando no fim à cidade inicial. Trata-se de um problema de dificuldade NP (dificuldade de tempo-polinomial não determinístico), cuja solução pretende diminuir o tempo gasto em uma determinada viagem no contexto real bem como os possíveis custos associados.

Formulação do problema

Formulação do problema

Existem diversos métodos para resolver o Problema do Caixeiro Viajante. O mais genérico é aquele que calcula todas as rotas possíveis, tendo em conta as características do problema, e escolhe a melhor de todas. Ilustrando, tome-se como por exemplo o seguinte grafo, em que cada vértice representa uma cidade diferente e onde cada aresta que liga duas cidades diferentes tem uma distância (ou custo) associado:

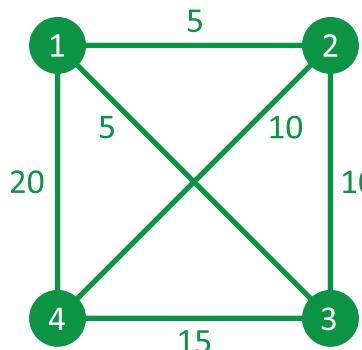


Figura 1 - Grafo

Assim, pelo método de **força bruta**, tem-se que:

1. Defina-se a cidade número 1 como a cidade inicial e consequentemente, também a final.
2. Calcule-se todos os caminhos possíveis começando e acabando na cidade número 1, passando uma única vez nas cidades números 2, 3 e 4.
3. Comparando todos os caminhos encontrados na alínea anterior, escolhe-se aquele com o menor custo.



Tendo em conta o ponto 3, sabe-se que o caminho de menor custo é o seguinte:



Formulação do problema

O problema do caixeiro viajante, quando resolvido através do método de força bruta, é considerado um dos problemas mais complexos a nível computacional. Visto que são calculados todos os itinerários possíveis dentro de um conjunto de cidades, a complexidade computacional deste algoritmo é de

$$O(n!)$$

Isto comprova-se pois, supondo que queremos calcular todos os itinerários possíveis num conjunto de 10 cidades, teremos o seguinte número de possibilidades:

$$n = 10$$

$$\text{possibilidades} = n * (n-1) * (n-2) \dots (n-(n-1))$$

$$\text{possibilidades} = 10 * 9 * 8 * 7 * 6 * 5 * 4 * 3 * 2 * 1 = n!$$

Ainda que este método seja suficiente para encontrar o melhor trajeto para um número pequeno de cidades, ele é ineficiente para um conjunto muito grande, visto que a partir de $n = 16$ cidades, computadores com processadores normais demoram cerca de 8 horas a calcular todos os trajetos possíveis. Para $n = 17$, demora cerca de $17 * 8 = 136$ horas e por aí adiante. Portanto, existe outro método mais eficaz que será explicado a seguir, que reduz a complexidade computacional para $O(n^2 \cdot 2^n)$!

Método de Programação Dinâmica

A técnica de programação dinâmica consiste em resolver um determinando problema combinado as soluções de sub-problemas mais pequenos recursivamente, ganhando eficiência no tempo gasto para os resolver, tendo o inconveniente de ocupar muita memória, pois cada resultado encontrado é guardado para ser reutilizado mais tarde.

Esta técnica é normalmente utilizada para resolver problemas de otimização, como é o caso do [Problema do Caixeiro Viajante](#), em que a solução para um problema de ordem $n!$ pode ser encontrada combinando as soluções de um ou mais problemas de ordem $(n-1)!$, que por sua vez pode conter soluções de ordem $(n-2)!$, e por aí adiante.

Formulação do problema

Para o caso em concreto do **Problema do Caixeiro Viajante** utilizando a técnica de programação dinâmica, considere-se novamente o grafo da figura 1 apresentado na página 4. Considerando novamente a cidade número 1 como a cidade final e inicial, devemos achar, para qualquer outro vértice i pertencente ao conjunto $\{2,3,4\}$, o melhor caminho começando em 1 e acabando em i e com todos os outros vértices aparecendo apenas uma única vez. Considere-se que o custo deste caminho é $custo(i)$. Então, o custo do ciclo correspondente será $custo(i) + distancia(i,1)$, em que $distancia(i,1)$ é a distância de i à cidade inicial. Ao calcular o custo de cada um dos melhores trajetos para i pertencente ao conjunto $\{2,3,4\}$, é retornando o melhor caminho.

Para calcular o $custo(i)$ utilizando a técnica de programação dinâmica, é necessário ter uma relação recursiva para resolver cada um dos sub-problemas. Defina-se $g(i,S)$ como o custo do melhor caminho visitando cada vértice do conjunto S apenas uma vez, começando no vértice 1 e acabando no vértice i . É portanto necessário começar em todos os subconjuntos de tamanho 2 e calcular $g(i,S)$, de seguida em todos os subconjuntos de tamanho 3 e calcular $g(i,S)$, e por aí adiante. Ilustrando este explicação, tem-se que:

$$\begin{aligned} g(1,\{2,3,4\}) = \min\{ & g(2,\{3,4\}) + distancia(2,1), \\ & g(3,\{2,4\}) + distancia(3,1), \\ & g(4,\{2,3\}) + distancia(4,1) \} \end{aligned}$$

Utilizando a relação de recorrência anterior, podemos encontrar soluções baseadas em programação dinâmica, resolvendo assim o **Problema do Caixeiro Viajante**. No pior caso, existem no máximo $O(n \cdot 2^n)$ sub-problemas, onde cada um demora um tempo linear a resolver. Portanto, a complexidade computacional deste método é de:

$$O(n^2 \cdot 2^n)$$

Aplicação do problema à programação

Método de caminhos aleatórios

Apesar de estes dois métodos serem suficientes para resolver o Problema do Caixeiro Viajante, decidimos utilizar também um algoritmo probabilístico que calcula um trajeto aleatório do conjunto de cidades dado e, de um conjunto de 20000000 caminhos aleatórios, escolhe o melhor. No entanto, à medida que o número de cidades vai aumentando, este algoritmo deixa de ser fiável pois, por exemplo, para $n = 12$, existem $12! = 479001600$ trajetos possíveis, um número muito superior aos 20000000 caminhos aleatórios gerados.

Aplicação do problema à programação

Para resolver o Problema do Caixeiro Viajante no contexto da unidade curricular de Algoritmos e Estruturas de Dados, os três métodos abordados anteriormente foram implementados utilizando a linguagem de programação C. Utilizando código previamente fornecido a todos os alunos, com informações sobre todas as distâncias entre cada um dos 18 distritos de Portugal Continental, o nosso grupo alterou-o de modo a ser possível calcular os melhores e piores trajetos desde 3 até 18 cidades, bem como a respetiva distância total, e medir os tempos de execução do processador para cada um dos três métodos descritos anteriormente. No entanto, estes caminhos diferem consoante os números mecanográficos dos alunos que realizaram este trabalho, bem como os tempos de execução, visto que os resultados foram obtidos utilizando dois computadores diferentes.

O código produzido encontra-se disponível em anexo no final deste documento.

Análise de resultados

O principais resultados pretendidos com os algoritmos implementados são:

- O melhor caminho para 3 até 16 cidades (valor da distância e conjunto de cidades percorridas)
- O pior caminho para 3 até 16 cidades (valor da distância e conjunto de cidades percorridas)
- Os tempos de execução para 3 até 16 cidades
- O histograma do número de ocorrências das distâncias de cada trajeto para 12 e 15 cidades
- O melhor e pior caminho desde 3 até 16 cidades utilizando 20000000 caminhos aleatórios
- Mapas dos melhores e piores caminhos encontrados

Por fim, era necessário obter novamente todos estes resultados mas para um valor de distâncias assimétrico, ou seja, onde a distância de Aveiro para Viseu seja diferente da distância de Viseu para Aveiro, por exemplo. Para tal, era apenas necessário alterar o valor da **variável *special* para *special* = 1**.

Todos os testes foram realizados para ambos os números mecanográficos 88808 e 88886 (João Miguel Nunes de Medeiros e Vasconcelos e Tiago Carvalho Mendes, respetivamente).

Output

Após implementarmos os três métodos analisados anteriormente (Força Bruta, Programação Dinâmica e Permutações Aleatórias), estava na altura de testar o código produzido. De seguida, seguem-se os resultados obtidos para ambos os números mecanográficos e com a variável *special* assumindo os valores 0 ou 1.

Força bruta | 88808 | Special = 0

Number of cities (n)	Execution time (s)	Melhor distância minPath	Pior distância maxPath
3	0,0001	883 [0,1,2]	883 [0,2,1]
4	0,0001	885 [0,1,3,2]	1402 [0,2,1,3]
5	0,0001	1085 [0,1,4,3,2]	1633 [0,3,1,2,4]
6	0,0001	1102 [0,1,2,3,4,5]	1836 [0,3,5,2,1,4]
7	0,0001	1101 [0,1,3,2,6,4,5]	2156 [0,3,5,2,4,1,6]
8	0,0001	1367 [0,1,7,4,5,6,3,2]	2680 [0,4,1,2,5,3,7,6]
9	0,001	1436 [0,1,7,5,4,8,2,3,6]	3312 [0,6,7,3,5,2,4,1,8]
10	0,01	1431 [0,1,9,7,5,4,8,2,3,6]	3663 [0,7,6,9,3,5,2,4,1,8]
11	0,061	1447 [0,1,9,7,5,4,8,2,3,6,10]	3942 [0,3,5,2,9,6,7,10,4,1,8]
12	0,669	1447 [0,1,9,7,5,4,11,8,2,3,6,10]	4381 [0,3,5,2,4,10,7,6,9,8,1,11]
13	7,893	1573 [0,1,9,7,5,4,11,8,12,2,3,6,10]	5003 [0,8,1,12,9,6,7,3,5,2,4,10,11]
14	106,519	1559 [0,1,9,7,4,11,8,12,2,3,6,10,13,5]	5136 [0,3,5,2,4,10,7,6,9,8,1,12,13,11]
15	2103,848	1575 [0,1,14,9,7,4,11,8,12,2,3,6,10,13,5]	5556 [0,8,1,12,14,6,9,3,5,2,4,10,7,13,11]
16	33934,154	1828 [0,1,14,9,7,4,11,8,12,15,2,3,6,10,13,5]	6475 [0,3,5,15,9,2,4,10,7,6,14,8,1,12,13,11]

Permutações aleatórias | 88808 | Special = 0

Number of cities (n)	Execution time (s)	minLength minPath	maxLength maxPath
3	0,712	883 [1,0,2]	883 [1,2,0]
4	0,923	885 [3,1,0,2]	1402 [3,0,2,1]
5	1,245	1085 [2,3,4,1,0]	1633 [0,3,1,2,4]
6	1,466	1102 [1,2,3,4,5,0]	1836 [3,0,4,1,2,5]
7	1,715	1101 [4,6,2,3,1,0,5]	2156 [6,0,3,5,2,4,1]
8	1,952	1367 [6,5,4,7,1,0,3,2]	2680 [1,2,5,3,7,6,0,4]
9	2,195	1436 [7,5,4,8,2,3,6,0,1]	3312 [0,6,7,3,5,2,4,1,8]
10	2,494	1431 [8,2,3,6,0,1,9,7,5,4]	3663 [1,4,2,5,3,9,6,7,0,8]
11	2,717	1447 [6,3,2,8,4,5,7,9,1,0,10]	3942 [7,6,9,2,5,3,0,8,1,4,10]
12	2,983	1447 [7,5,4,11,8,2,3,6,10,0,1,9]	4381 [2,5,3,9,8,1,11,0,6,7,10,4]
13	3,185	1600 [2,12,8,11,4,7,9,5,0,1,10,6,3]	5003 [7,6,9,8,1,12,0,11,10,4,2,5,3]
14	3,498	1756 [4,7,9,1,0,5,13,6,10,2,3,12,8,11]	5131 [11,13,8,1,12,9,2,4,10,5,3,7,6,0]
15	3,73	1811 [11,12,8,2,3,10,5,4,7,9,14,1,0,13,6]	5522 [2,5,6,7,13,11,14,8,0,12,1,3,4,10,9]
16	3,946	2170 [9,4,5,0,13,11,15,12,8,2,3,6,10,1,14,7]	6430 [12,0,8,1,3,4,10,7,6,5,15,9,2,14,11,13]
17	4,264	2376 [13,9,7,5,14,1,0,10,11,8,12,15,2,3,6,16,4]	6656 [12,0,8,14,15,9,10,4,2,5,3,7,6,11,13,16,1]
18	4,491	2521 [13,10,6,15,12,2,3,8,11,4,5,14,17,1,0,9,7,16]	6969 [2,0,8,14,3,17,16,1,12,9,15,5,11,13,10,4,6,7]

Programação Dinâmica | 88808 | Special = 0

Number of cities (n)	Execution time (s)	minLength minPath	maxLength maxPath
3	0,018	883 [0,1,2]	883 [0,1,2]
4	0,014	885 [0,1,3,2]	1402 [0,2,1,3]
5	0,011	1085 [0,1,4,3,2]	1633 [0,3,1,2,4]
6	0,012	1102 [0,1,2,3,4,5]	1836 [0,3,5,2,1,4]
7	0,011	1101 [0,1,3,2,6,4,5]	2156 [0,3,5,2,4,1,6]
8	0,011	1367 [0,1,7,4,5,6,2,3]	2680 [0,4,1,2,5,3,7,6]
9	0,014	1436 [0,1,7,5,4,8,2,3,6]	3312 [0,6,7,3,5,2,4,1,8]
10	0,014	1431 [0,1,9,7,5,4,8,2,3,6]	3663 [0,7,6,9,3,5,2,4,1,8]
11	0,014	1447 [0,1,9,7,5,4,8,2,3,6,10]	3942 [0,3,5,2,9,6,7,10,4,1,8]
12	0,012	1447 [0,1,9,7,5,4,11,8,2,3,6,10]	4381 [0,3,5,2,4,10,7,6,9,8,1,11]
13	0,017	1573 [0,1,9,7,5,4,11,8,12,2,3,6,10]	5003 [0,8,1,12,9,6,7,3,5,2,4,10,11]
14	0,019	1559 [0,1,9,7,4,11,8,12,2,3,6,10,13,5]	5136 [0,3,5,2,4,10,7,6,9,8,1,12,13,11]
15	0,033	1575 [0,1,14,9,7,4,11,8,12,2,3,6,10,13,5]	5556 [0,8,1,12,14,6,9,3,5,2,4,10,7,13,11]
16	0,066	1828 [0,1,14,9,7,4,11,8,12,15,2,3,6,10,13,5]	6475 [0,3,5,15,9,2,4,10,7,6,14,8,1,12,13,11]
17	0,128	1832 [0,1,14,9,7,4,16,11,8,12,15,2,3,6,10,13,5]	6798 [0,8,1,16,10,4,2,5,15,9,3,7,6,14,11,13,12]
18	0,266	1839 [0,5,13,10,6,3,2,15,12,8,11,16,4,7,9,14,1,17]	7122 [0,3,5,15,9,2,4,10,7,6,14,8,1,16,13,11,17,12]

Análise de Resultados | Output

Força bruta | 88808 | Special = 1

Number of cities (n)	Execution time (s)	Melhor distância	minPath	Pior distância	maxPath
3	0,0001	764	[0,2,1]	1002	[0,1,2]
4	0,0001	775	[0,3,2,1]	1417	[0,3,1,2]
5	0,0001	934	[0,4,3,2,1]	1716	[0,2,4,1,3]
6	0,0001	971	[0,5,4,3,2,1]	1986	[0,5,2,4,1,3]
7	0,0001	988	[0,5,4,6,2,3,1]	2290	[0,6,5,2,4,1,3]
8	0,0001	1260	[0,6,2,3,1,7,4,5]	2888	[0,6,5,2,4,1,3,7]
9	0,001	1362	[0,6,3,2,8,4,5,7,1]	3482	[0,3,7,6,5,2,4,1,8]
10	0,008	1314	[0,8,2,3,6,1,9,7,4,5]	3901	[0,5,3,7,6,9,2,4,1,8]
11	0,067	1343	[0,10,3,2,8,6,1,9,7,4,5]	4114	[0,6,9,3,5,2,7,10,4,1,8]
12	0,709	1347	[0,1,9,7,4,11,8,2,3,6,10,5]	4484	[0,11,1,3,7,6,5,2,4,10,9,8]
13	8,679	1479	[0,1,9,7,4,11,8,12,2,3,6,10,5]	5202	[0,6,9,2,4,8,1,11,10,5,3,7,12]
14	117,306	1481	[0,1,9,7,4,11,8,12,2,3,6,10,13,5]	5412	[0,11,1,8,13,6,9,2,4,10,5,3,7,12]
15	1780,078	1520	[0,1,14,9,7,4,11,8,12,2,3,6,10,13,5]	5925	[0,11,13,9,2,4,10,5,3,7,6,1,8,14,12]
16	29231,248	1722	[0,1,14,9,7,4,11,8,12,15,2,3,6,10,13,5]	6863	[0,3,7,6,5,2,4,10,9,15,13,11,1,8,14,12]

Permutações aleatórias | 88808 | Special = 1

Number of cities (n)	Execution time (s)	minLength	minPath	maxLength	maxPath
3	0,693	764	[1,0,2]	1002	[1,2,0]
4	0,964	775	[2,1,0,3]	1417	[0,3,1,2]
5	1,241	934	[4,3,2,1,0]	1716	[0,2,4,1,3]
6	1,507	971	[3,2,1,0,5,4]	1986	[0,5,2,4,1,3]
7	1,849	988	[4,6,2,3,1,0,5]	2290	[4,1,3,0,6,5,2]
8	2,113	1260	[4,5,0,6,2,3,1,7]	2888	[6,5,2,4,1,3,7,0]
9	2,365	1362	[4,5,7,1,0,6,3,2,8]	3482	[6,5,2,4,1,8,0,3,7]
10	2,7	1314	[2,3,6,1,9,7,4,5,0,8]	3901	[2,4,1,8,0,5,3,7,6,9]
11	2,954	1343	[2,8,6,1,9,7,4,5,0,10,3]	4114	[3,5,2,7,10,4,1,8,0,6,9]
12	3,244	1386	[7,5,4,11,8,2,3,6,10,0,1,9]	4482	[5,3,7,6,9,8,0,11,1,2,4,10]
13	3,489	1607	[5,0,1,9,7,4,11,8,2,12,3,6,10]	5182	[8,9,2,4,6,5,3,7,12,0,11,10,1]
14	3,792	1724	[11,6,3,2,12,8,10,13,0,1,5,9,7,4]	5307	[5,8,13,11,1,6,9,2,7,12,0,3,4,10]
15	4,083	1776	[3,6,10,13,1,9,7,4,5,0,14,11,8,12,2]	5796	[14,12,0,6,1,11,5,3,7,10,4,8,13,9,2]
16	4,312	2112	[9,4,5,0,13,11,15,12,8,2,3,6,10,1,14,7]	6767	[5,2,4,8,14,12,0,11,13,6,1,3,9,15,7,10]
17	4,612	2297	[13,9,7,5,14,1,0,10,11,8,12,15,2,3,6,16,4]	6919	[1,8,9,2,0,11,13,10,4,15,7,6,5,3,16,14,12]
18	4,913	2493	[14,1,13,10,2,12,8,15,3,6,16,11,0,17,5,4,7,9]	7328	[2,7,6,17,11,5,15,14,10,4,8,0,16,1,3,13,12,9]

Programação Dinâmica | 88808 | Special = 1

Number of cities (n)	Execution time (s)	minLength	minPath	maxLength	maxPath
3	0,022	764	[0,2,1]	1002	[0,1,2]
4	0,014	775	[0,3,2,1]	1417	[0,3,1,2]
5	0,011	934	[0,4,3,2,1]	1716	[0,2,4,1,3]
6	0,013	971	[0,5,4,3,2,1]	1986	[0,5,2,4,1,3]
7	0,012	988	[0,5,4,6,2,3,1]	2290	[0,6,5,2,4,1,3]
8	0,011	1260	[0,6,2,3,1,7,4,5]	2888	[0,6,5,2,4,1,3,7]
9	0,014	1362	[0,6,3,2,8,4,5,7,1]	3482	[0,3,7,6,5,2,4,1,8]
10	0,011	1314	[0,8,2,3,6,1,9,7,4,5]	3901	[0,5,3,7,6,9,2,4,1,8]
11	0,014	1343	[0,10,3,2,8,6,1,9,7,4,5]	4114	[0,6,9,3,5,2,7,10,4,1,8]
12	0,013	1347	[0,1,9,7,4,11,8,2,3,6,10,5]	4484	[0,11,1,3,7,6,5,2,4,10,9,8]
13	0,017	1479	[0,1,9,7,4,11,8,12,2,3,6,10,5]	5202	[0,6,9,2,4,8,1,11,10,5,3,7,12]
14	0,019	1481	[0,1,9,7,4,11,8,12,2,3,6,10,13,5]	5412	[0,11,1,8,13,6,9,2,4,10,5,3,7,12]
15	0,032	1520	[0,1,14,9,7,4,11,8,12,2,3,6,10,13,5]	5925	[0,11,13,9,2,4,10,5,3,7,6,1,8,14,12]
16	0,065	1722	[0,1,14,9,7,4,11,8,12,15,2,3,6,10,13,5]	6863	[0,3,7,6,5,2,4,10,9,15,13,11,1,8,14,12]
17	0,127	1763	[0,1,14,9,7,4,16,11,8,12,15,2,3,6,10,13,5]	7223	[0,11,10,5,3,7,6,9,2,4,15,13,16,1,8,14,12]
18	0,278	1769	[0,17,1,14,9,7,4,16,11,8,12,15,2,3,6,10,13,5]	7655	[0,3,5,2,4,10,9,15,7,6,17,11,13,16,1,8,14,12]

Análise de Resultados | Output

Força bruta | 88886 | Special = 0

Number of cities (n)	Execution time (s)	minLength	minPath	maxLength	maxPath
3	0.0001	683	[0,1,2]	683	[0,2,1]
4	0.0001	862	[0,1,3,2]	1335	[0,2,1,3]
5	0.0001	980	[0,1,3,2,4]	1497	[0,2,1,4,3]
6	0.0001	1125	[0,1,3,2,4,5]	1794	[0,2,1,5,3,4]
7	0.0001	1125	[0,1,6,3,2,4,5]	2335	[0,2,1,5,3,4,6]
8	0.0001	1137	[0,1,6,3,7,2,4,5]	2909	[0,2,1,7,5,3,4,6]
9	0.001	1153	[0,1,6,3,7,2,8,4,5]	2940	[0,2,1,7,5,3,4,6,8]
10	0.006	1279	[0,1,6,3,9,7,2,8,4,5]	3549	[0,2,1,7,5,3,4,6,8,9]
11	0.065	1336	[0,4,5,10,1,6,3,9,7,2,8]	4107	[0,2,1,7,10,8,6,4,9,5,3]
12	0.666	1333	[0,4,5,10,1,11,6,3,9,7,2,8]	4508	[0,2,1,7,11,8,10,3,5,9,4,6]
13	7,956	1568	[0,4,5,10,12,1,11,6,3,9,7,2,8]	5044	[0,3,5,9,10,7,1,2,12,8,11,4,6]
14	116,675	1572	[0,4,5,10,12,1,11,13,6,3,9,7,2,8]	5461	[0,3,1,2,12,8,11,7,10,9,5,13,4,6]
15	2009,91	1584	[0,4,5,10,12,1,11,13,6,3,9,7,2,14,8]	5680	[0,3,5,13,8,11,7,1,2,12,14,10,9,4,6]
16	33068,312	1570	[0,1,15,8,14,2,7,9,3,6,13,11,12,10,5,4]	5783	[0,2,1,7,11,8,12,14,10,3,4,6,15,9,5,13]

Permutações aleatórias | 88886 | Special = 0

Number of cities (n)	Execution time (s)	Melhor distância	minPath	Pior distância	maxPath
3	0,698	683	[0,1,2]	683	[1,2,0]
4	0,921	862	[2,0,1,3]	1335	[3,0,2,1]
5	1,181	980	[3,1,0,4,2]	1497	[0,3,4,1,2]
6	1,532	1125	[3,2,4,5,0,1]	1794	[5,1,2,0,4,3]
7	1,709	1125	[1,6,3,2,4,5,0]	2335	[4,3,5,1,2,0,6]
8	1,989	1137	[4,2,7,3,6,1,0,5]	2909	[4,6,0,2,1,7,5,3]
9	2,237	1153	[0,1,6,3,7,2,8,4,5]	2940	[6,4,3,5,7,1,2,0,8]
10	2,531	1279	[4,5,0,1,6,3,9,7,2,8]	3549	[3,0,2,1,7,5,9,4,6,8]
11	2,732	1336	[7,2,8,0,4,5,10,1,6,3,9]	4107	[3,0,2,10,7,1,8,6,4,9,5]
12	3,013	1333	[5,10,1,11,6,3,9,7,2,8,0,4]	4508	[2,1,7,11,8,0,6,4,3,5,9,10]
13	3,214	1568	[9,7,2,8,0,4,5,10,12,1,11,6,3]	5044	[2,12,8,11,4,6,0,9,5,3,10,7,1]
14	3,473	1648	[10,12,5,4,0,8,7,2,9,3,6,13,11,1]	5419	[3,4,13,5,9,1,2,12,7,11,8,10,6,0]
15	4,055	1816	[6,9,3,2,7,14,8,0,5,12,10,4,1,11,13]	5646	[8,13,5,6,0,3,4,9,1,7,10,2,12,14,11]
16	5,228	1943	[10,12,1,11,15,8,13,6,3,9,2,7,14,0,4,5]	5707	[5,6,15,9,1,7,12,8,11,14,10,3,0,2,4,13]
17	4,225	2331	[13,15,12,10,1,11,0,4,5,8,14,7,2,16,9,3,6]	6523	[8,1,16,4,6,0,3,10,2,13,5,9,15,14,12,7,11]
18	4,489	2424	[17,5,1,11,12,10,0,13,6,3,7,2,9,16,14,8,15,4]	7004	[14,15,6,0,13,5,3,4,9,1,2,10,16,12,7,11,8,17]

Programação Dinâmica | 88886 | Special = 0

Number of cities (n)	Execution time (s)	minLength	minPath	maxLength	maxPath
3	0,034	683	[0,1,2]	683	[0,1,2]
4	0,013	862	[0,1,3,2]	1335	[0,2,1,3]
5	0,015	980	[0,1,3,2,4]	1497	[0,2,1,4,3]
6	0,015	1125	[0,1,3,2,4,5]	1794	[0,2,1,5,3,4]
7	0,016	1125	[0,1,6,3,2,4,5]	2335	[0,2,1,5,3,4,6]
8	0,012	1137	[0,1,6,3,7,2,4,5]	2909	[0,2,1,7,5,3,4,6]
9	0,015	1153	[0,1,6,3,7,2,8,4,5]	2940	[0,2,1,7,5,3,4,6,8]
10	0,012	1279	[0,1,6,3,9,7,2,8,4,5]	3549	[0,2,1,7,5,3,4,6,8,9]
11	0,013	1336	[0,4,5,10,1,6,3,9,7,2,8]	4107	[0,2,1,7,10,8,6,4,3,5,9]
12	0,013	1333	[0,4,5,10,1,11,6,3,9,7,2,8]	4508	[0,2,1,7,11,8,10,3,5,9,4,6]
13	0,018	1568	[0,4,5,10,12,1,11,6,3,9,7,2,8]	5044	[0,3,5,9,10,7,1,2,12,8,11,4,6]
14	0,019	1572	[0,4,5,10,12,1,11,13,6,3,9,7,2,8]	5461	[0,3,1,2,12,8,11,7,10,9,5,13,4,6]
15	0,033	1584	[0,4,5,10,12,1,11,13,6,3,9,7,2,14,8]	5680	[0,3,5,13,8,11,7,1,2,12,14,10,9,4,6]
16	0,062	1570	[0,1,15,8,14,2,7,9,3,6,13,11,12,10,5,4]	5783	[0,2,1,7,11,8,12,14,10,3,4,6,15,9,5,13]
17	0,124	1823	[0,1,15,8,14,2,7,16,9,3,6,13,11,12,10,5,4]	6683	[0,2,12,14,10,16,1,7,11,8,13,5,3,4,6,15,9]
18	0,27	1839	[0,1,15,8,14,2,7,16,9,3,6,13,11,12,10,17,5,4]	7122	[0,2,1,16,10,7,11,8,12,14,17,3,4,6,15,9,5,13]

Análise de Resultados | Output

Força bruta | 88886 | Special = 1

Number of cities (n)	Execution time (s)	Melhor distância	minPath	Pior distância	maxPath
3	0,0001	580	[0,2,1]	786	[0,1,2]
4	0,0001	755	[0,3,2,1]	1362	[0,2,1,3]
5	0,0001	851	[0,3,2,4,1]	1632	[0,1,2,4,3]
6	0,0001	995	[0,3,2,4,5,1]	1989	[0,4,2,1,5,3]
7	0,0001	1060	[0,6,3,2,4,5,1]	2434	[0,1,2,4,6,5,3]
8	0,0001	1060	[0,2,7,3,6,1,4,5]	3075	[0,2,1,7,4,6,5,3]
9	0,001	1073	[0,8,2,7,3,6,1,4,5]	3170	[0,2,1,7,4,6,8,5,3]
10	0,007	1193	[0,8,2,7,9,3,6,1,4,5]	3816	[0,2,1,7,4,6,8,3,5,9]
11	0,067	1216	[0,8,2,7,9,3,6,4,5,10,1]	4361	[0,6,8,1,2,10,7,4,3,5,9]
12	0,655	1305	[0,4,5,10,1,11,6,3,9,7,2,8]	4721	[0,6,5,3,10,2,11,8,1,7,4,9]
13	8,007	1485	[0,8,2,7,9,3,6,4,5,10,12,11,1]	5359	[0,8,11,5,3,1,7,4,6,10,2,12,9]
14	150,364	1508	[0,4,5,10,12,8,2,7,9,3,6,13,11,1]	5703	[0,6,5,13,4,3,10,7,11,8,1,2,12,9]
15	2218,402	1526	[0,4,5,10,12,11,13,6,3,9,7,2,14,8,1]	5960	[0,14,4,6,10,7,11,8,13,5,3,1,2,12,9]
16	32859,267	1528	[0,4,5,10,12,11,13,6,3,9,7,2,14,8,15,1]	6188	[0,14,4,6,15,13,5,3,10,7,11,8,1,2,12,9]

Permutações aleatórias | 88886 | Special = 1

Number of cities (n)	Execution time (s)	minLength	minPath	maxLength	maxPath
3	0,022	580	[0,2,1]	786	[0,1,2]
4	0,014	755	[0,3,2,1]	1362	[0,2,1,3]
5	0,011	851	[0,3,2,4,1]	1632	[0,1,2,4,3]
6	0,013	995	[0,3,2,4,5,1]	1989	[0,4,2,1,5,3]
7	0,014	1060	[0,6,3,2,4,5,1]	2434	[0,1,2,4,6,5,3]
8	0,013	1060	[0,2,7,3,6,1,4,5]	3075	[0,2,1,7,4,6,5,3]
9	0,011	1073	[0,8,2,7,3,6,1,4,5]	3170	[0,2,1,7,4,6,8,5,3]
10	0,011	1193	[0,8,2,7,9,3,6,1,4,5]	3816	[0,2,1,7,4,6,8,3,5,9]
11	0,014	1216	[0,8,2,7,9,3,6,4,5,10,1]	4361	[0,6,8,1,2,10,7,4,3,5,9]
12	0,012	1305	[0,4,5,10,1,11,6,3,9,7,2,8]	4721	[0,6,5,3,10,2,11,8,1,7,4,9]
13	0,018	1485	[0,8,2,7,9,3,6,4,5,10,12,11,1]	5359	[0,8,11,5,3,1,7,4,6,10,2,12,9]
14	0,022	1508	[0,4,5,10,12,8,2,7,9,3,6,13,11,1]	5703	[0,6,5,13,4,3,10,7,11,8,1,2,12,9]
15	0,033	1526	[0,4,5,10,12,11,13,6,3,9,7,2,14,8,1]	5960	[0,14,4,6,1,7,11,8,13,5,3,10,2,12,9]
16	0,062	1528	[0,4,5,10,12,11,13,6,3,9,7,2,14,8,15,1]	6188	[0,14,4,6,15,13,5,3,10,7,11,8,1,2,12,9]
17	0,135	1730	[0,4,5,10,12,11,13,6,3,9,16,7,2,14,8,15,1]	7118	[0,14,10,2,1,7,4,3,15,13,5,6,8,11,16,12,9]
18	0,275	1769	[0,4,5,17,10,12,11,13,6,3,9,16,7,2,14,8,15,1]	7655	[0,2,1,7,11,8,10,16,12,14,4,6,15,13,5,3,17,9]

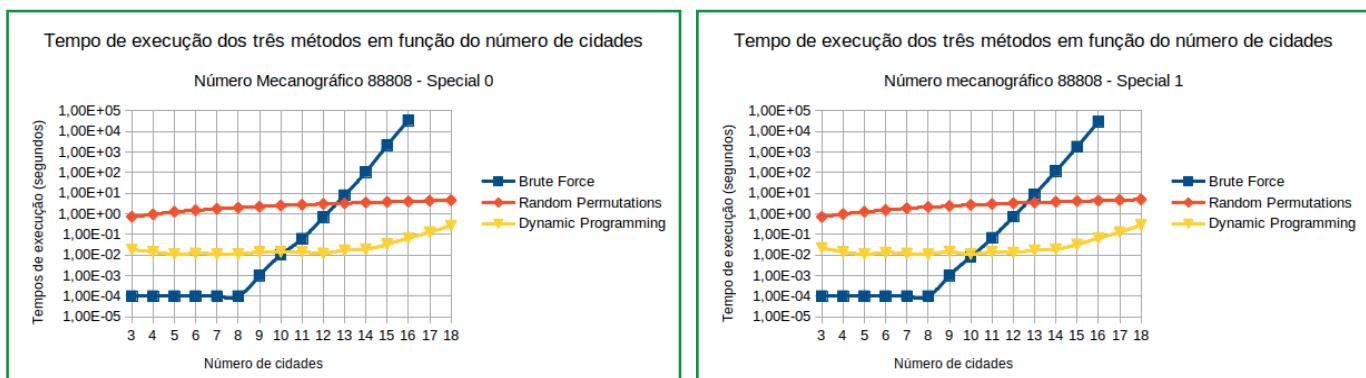
Programação Dinâmica | 88886 | Special = 1

Number of cities (n)	Execution time (s)	minLength	minPath	maxLength	maxPath
3	0,725	580	[1,0,2]	786	[1,2,0]
4	0,981	755	[1,0,3,2]	1362	[3,0,2,1]
5	1,276	851	[1,0,3,2,4]	1632	[2,4,3,0,1]
6	1,512	995	[2,4,5,1,0,3]	1989	[4,2,1,5,0]
7	1,829	1060	[6,3,2,4,5,1,0]	2434	[0,1,2,4,6,5,3]
8	2,08	1060	[1,4,5,0,2,7,3,6]	3075	[2,1,7,4,6,5,3,0]
9	2,375	1073	[4,5,0,8,2,7,3,6,1]	3170	[4,6,8,5,3,0,2,1,7]
10	2,729	1193	[7,9,3,6,1,4,5,0,8,2]	3816	[7,4,6,8,3,5,9,0,2,1]
11	2,984	1216	[0,8,2,7,9,3,6,4,5,10,1]	4361	[2,10,7,4,3,5,9,0,6,8,1]
12	3,256	1305	[5,10,1,11,6,3,9,7,2,8,0,4]	4694	[9,0,6,1,2,11,8,10,7,4,3,5]
13	3,527	1541	[9,3,6,11,1,0,4,5,10,12,8,2,7]	5310	[12,9,0,8,11,5,6,10,3,1,7,4,2]
14	3,8	1594	[10,12,5,4,0,8,7,2,9,3,6,13,11,1]	5594	[6,0,13,5,3,10,2,1,7,11,8,12,9,4]
15	4,131	1828	[2,7,3,9,6,13,11,10,1,0,5,4,12,8,14]	5809	[3,10,14,4,6,5,2,12,9,0,13,8,1,7,11]
16	4,379	1873	[2,7,9,13,14,8,15,0,4,5,10,12,1,11,6,3]	5957	[13,5,7,10,8,11,3,12,9,0,2,1,14,4,6,15]
17	4,703	2362	[9,2,7,3,6,14,8,15,10,5,4,0,12,1,11,13,16]	6772	[15,9,0,13,5,7,4,16,12,2,8,11,14,1,3,10,6]
18	5,031	2330	[17,5,1,11,12,10,0,13,6,3,7,2,9,16,14,8,15,4]	7246	[13,17,9,10,16,12,14,15,2,11,3,5,8,1,7,4,6,0]

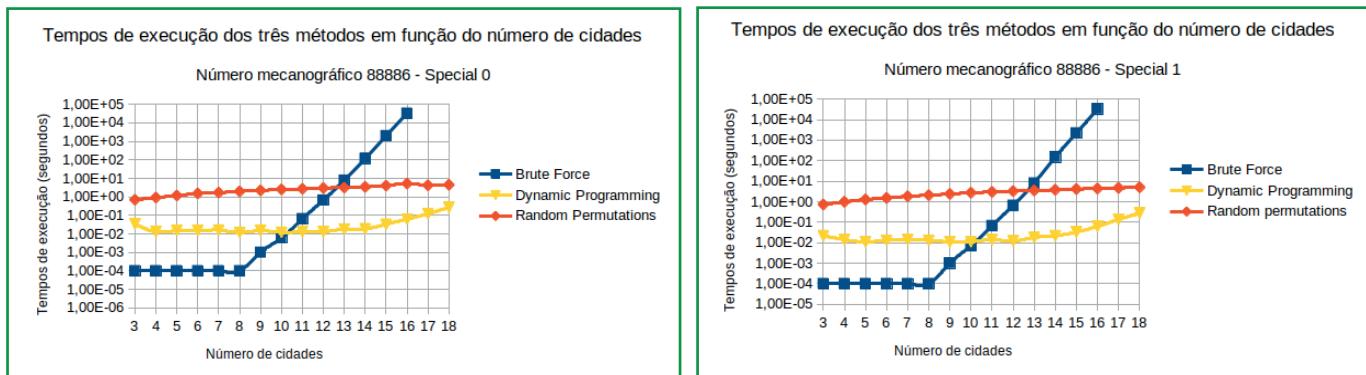
Tempos de execução

Tendo em conta os outputs mostrados anteriormente, decidimos também fazer gráficos dos tempos de execução dos três métodos utilizados em função do número de cidades. Concluímos que os tempos de execução para o método da força bruta crescem com complexidade $O(n!)$ e para o método de programação dinâmica com complexidade $O(n^2 \cdot 2^n)$, como demonstrados teoricamente e comprovados de seguida. No entanto, visto que os testes foram realizados, para cada número mecanográfico, no respetivo computador pessoal do aluno, e visto que ambos possuem processadores diferentes, os tempos de execução são também um pouco diferentes.

Número mecanográfico 88808



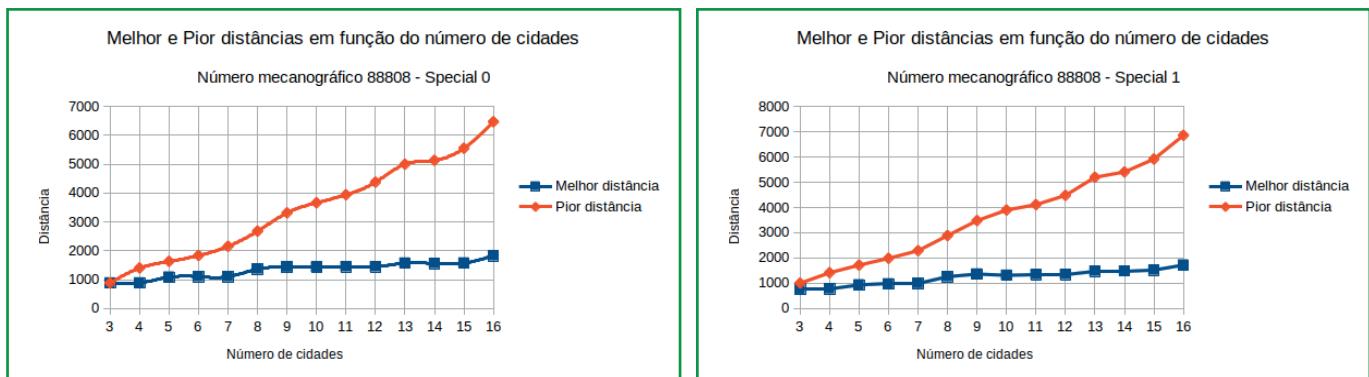
Número mecanográfico 88886



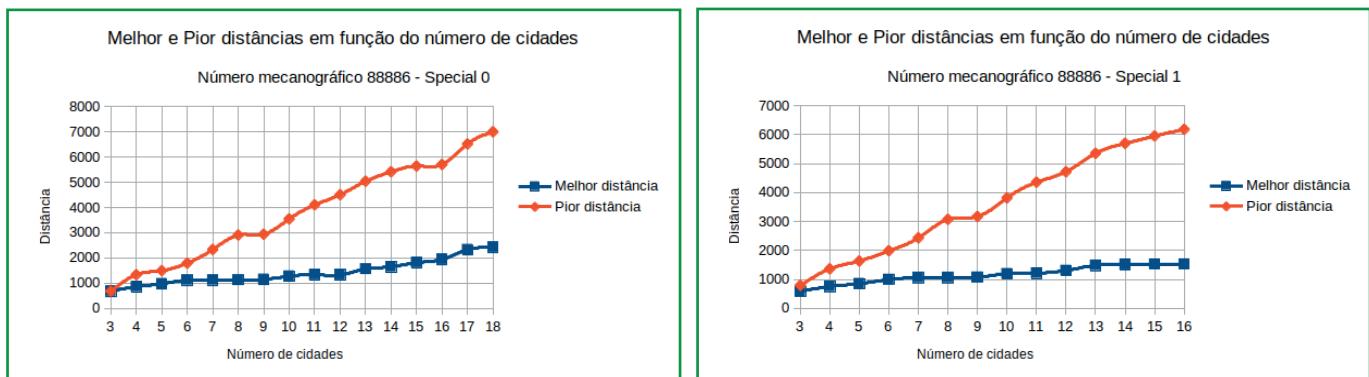
Variação das distâncias dos melhores e piores caminhos

Tendo em conta os outputs mostrados anteriormente, de seguida são mostrados gráficos da variação das melhores e piores distâncias para ambos os elementos do grupo, unicamente para o método de Força Bruta, visto que nos restantes métodos os gráficos seriam muito semelhantes. Pela análise dos gráficos, conclui-se que à medida que o número de cidades aumenta, as melhores e piores distâncias também aumentam, como era de esperar.

Número mecanográfico 88808



Número mecanográfico 88886

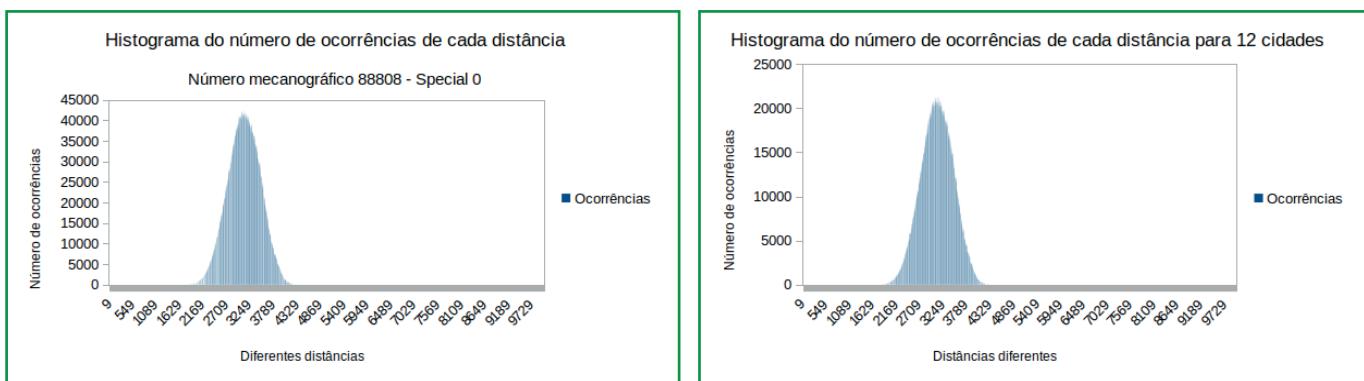


Análise de Resultados | Histogramas das diferentes distâncias

Histograma do número de ocorrência de cada diferente distância

Para uma análise mais aprofundada dos resultados obtidos, foram efetuados histogramas que demonstram o número de ocorrências de cada uma das diferentes distâncias calculadas para 12 e 15 cidades, para ambos os elementos do grupo e para os diferentes valores da variável **special**. Estes histogramas foram ambos realizados para os métodos de força bruta e de permutações aleatórias, sendo comparados de seguida. Nota: Os seguintes histogramas apenas se referem ao número mecanográfico 88808, visto que para ambos os números mecanográficos apresentam valores semelhantes.

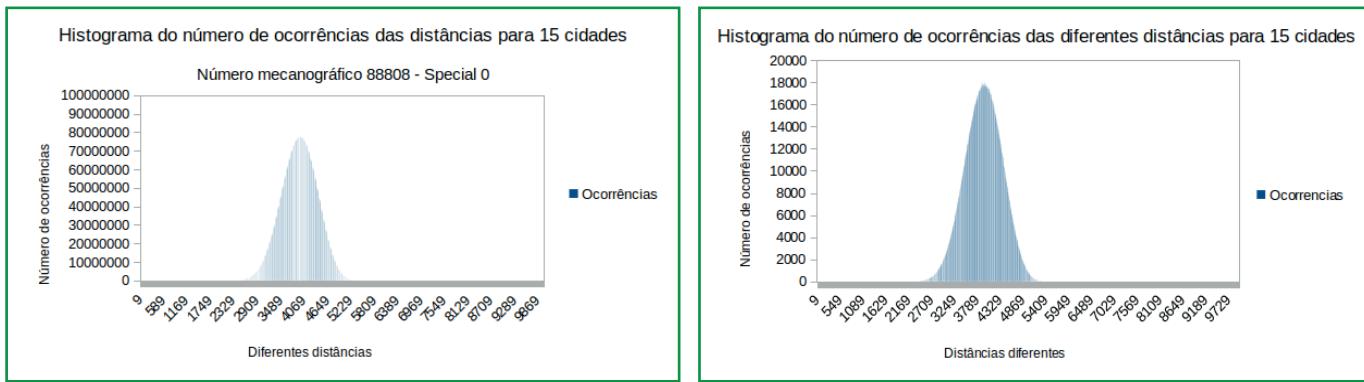
88808 - 12 cidades - Special = 0



Força bruta

Permutações aleatórias

88808 - 15 cidades - Special = 0

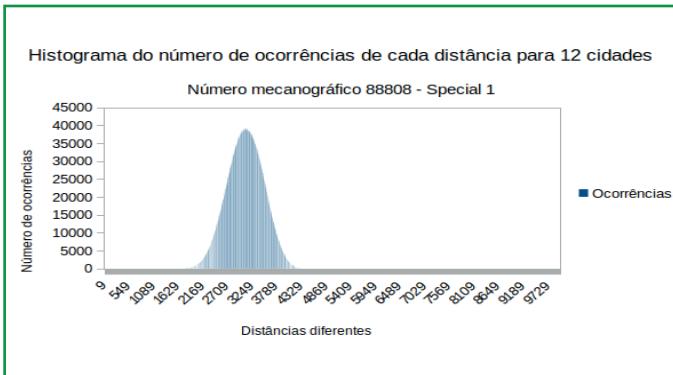


Força bruta

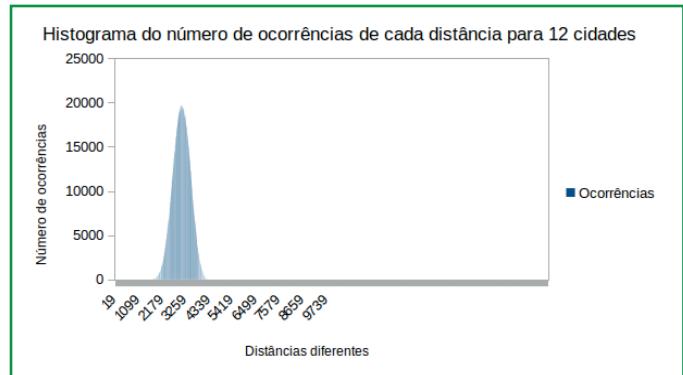
Permutações aleatórias

Análise de Resultados | Histogramas das diferentes distâncias

88808 - 12 cidades - Special = 1

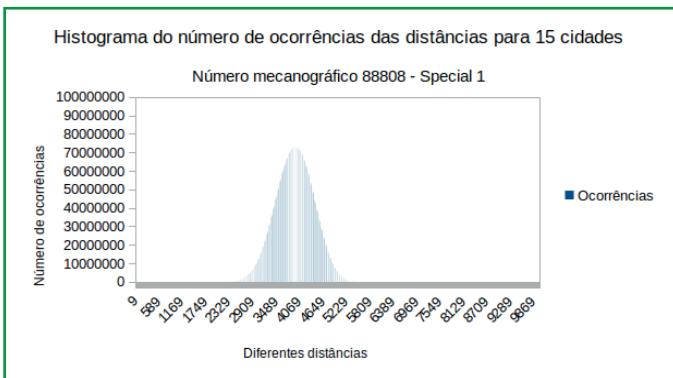


Força bruta

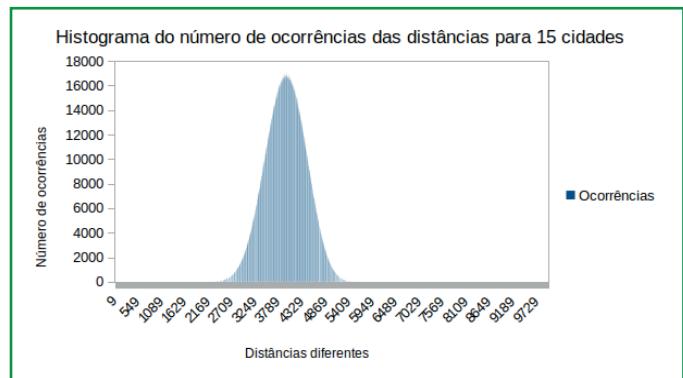


Permutações aleatórias

88808 - 15 cidades - Special = 1



Força bruta

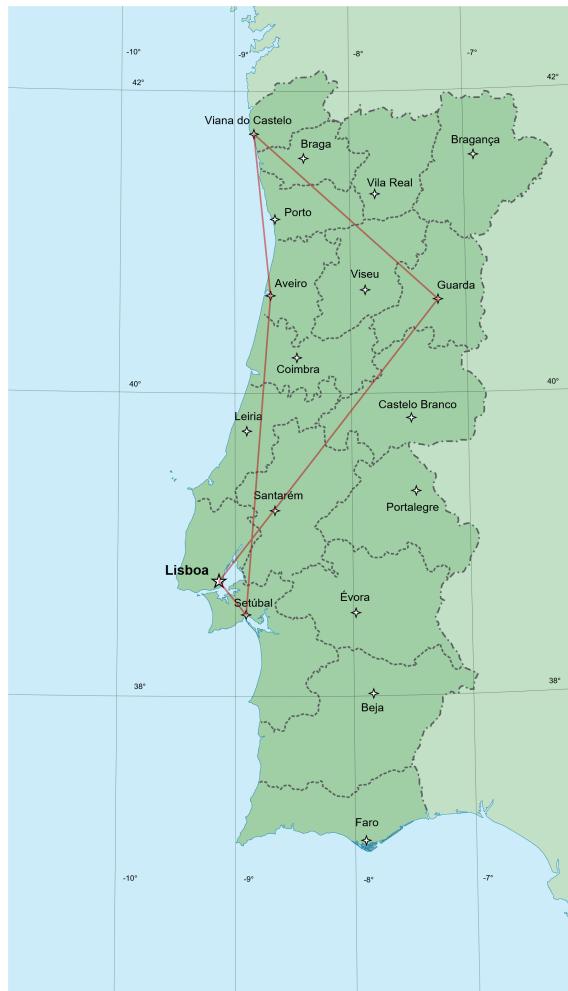


Permutações aleatórias

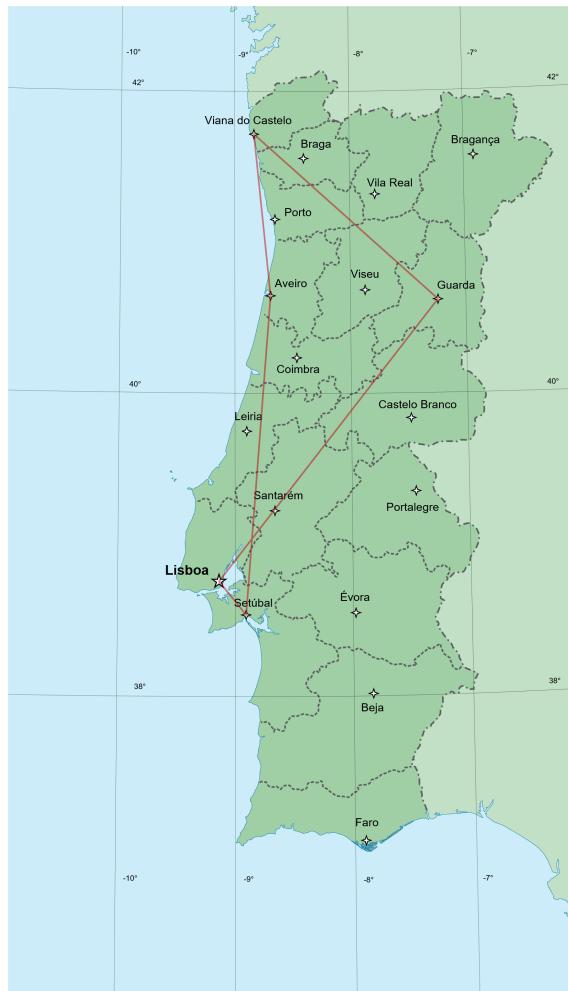
Mapas gerados

Ao efetuar os testes anteriores, usámos também o código já disponibilizado para a criação de grafos num ficheiro SVG, cujo fundo é uma imagem do mapa de Portugal Continental, ligando assim os diferentes distritos portugueses consoante o número mecanográfico utilizado. De seguida, seguem-se mapas dos melhores e piores trajetos para 5 e 18 cidades, comparados lado a lado os utilizados no método de programação dinâmica e de permutações aleatórias. No final, são comparados os melhores caminhos para $n = 5$ com $\text{special} = 0$ e $\text{special} = 1$ (nº mec 88808) e de seguida os melhores caminhos para $n = 5$ dos números mecanográfico 88808 e 88886 (special = 0).

88808 | Melhor trajeto para $n = 5$
Programação dinâmica



88808 | Melhor trajeto para $n = 5$
Permutações aleatórias



Análise de Resultados | Mapas gerados

88808 | Pior trajeto para $n = 5$
Programação dinâmica

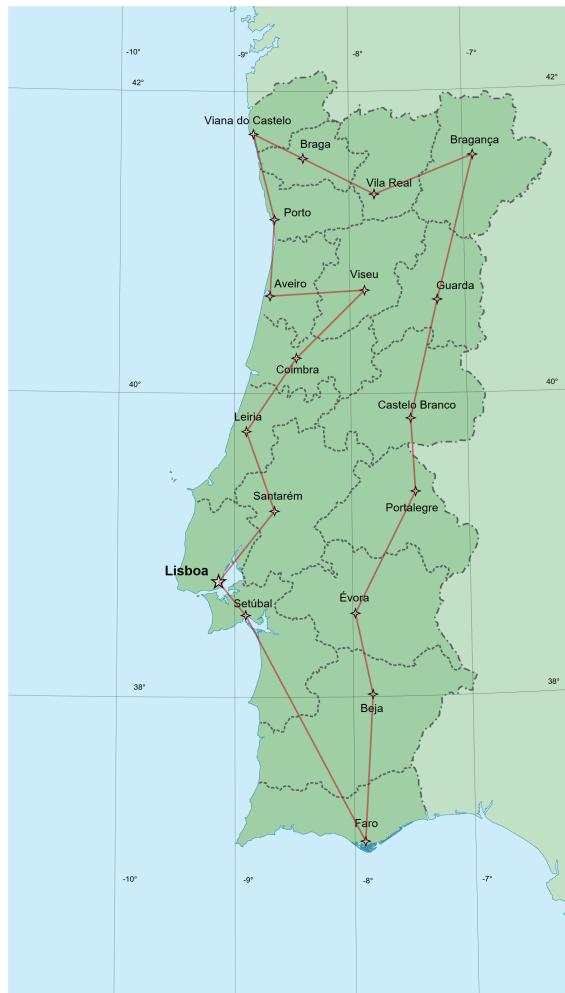


88808 | Pior trajeto para $n = 5$
Permutações aleatórias

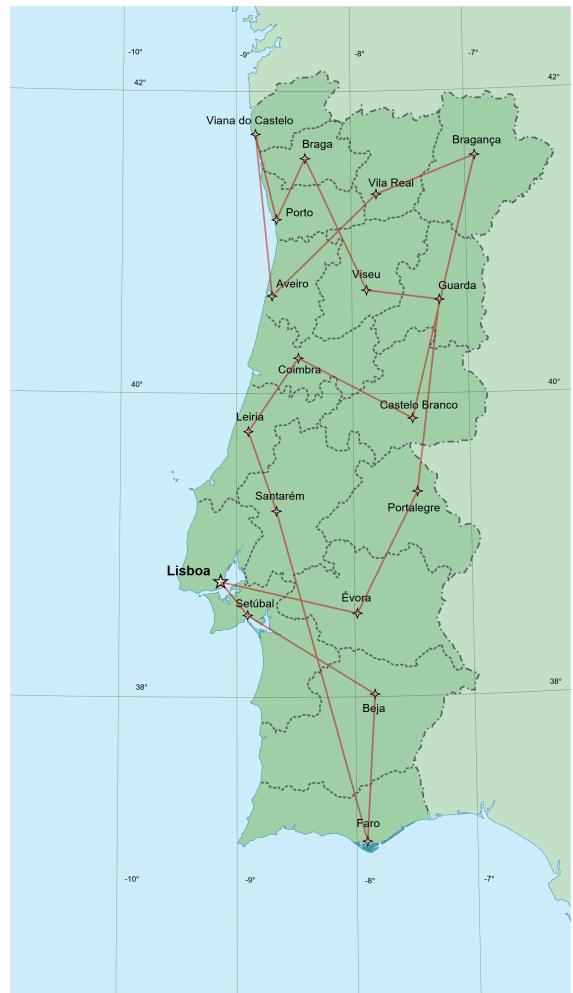


Análise de Resultados | Mapas gerados

88808 | Melhor trajeto para n = 18
Programação dinâmica



88808 | Melhor trajeto para n = 18
Permutações aleatórias



Análise de Resultados | Mapas gerados

88808 | Pior trajeto para $n = 18$
Programação dinâmica

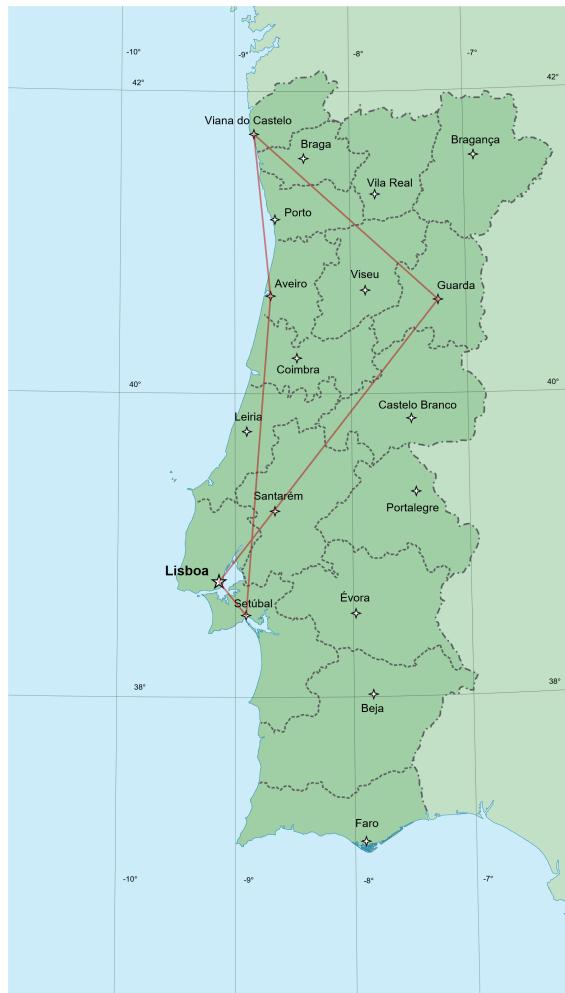


88808 | Pior trajeto para $n = 18$
Permutações aleatórias

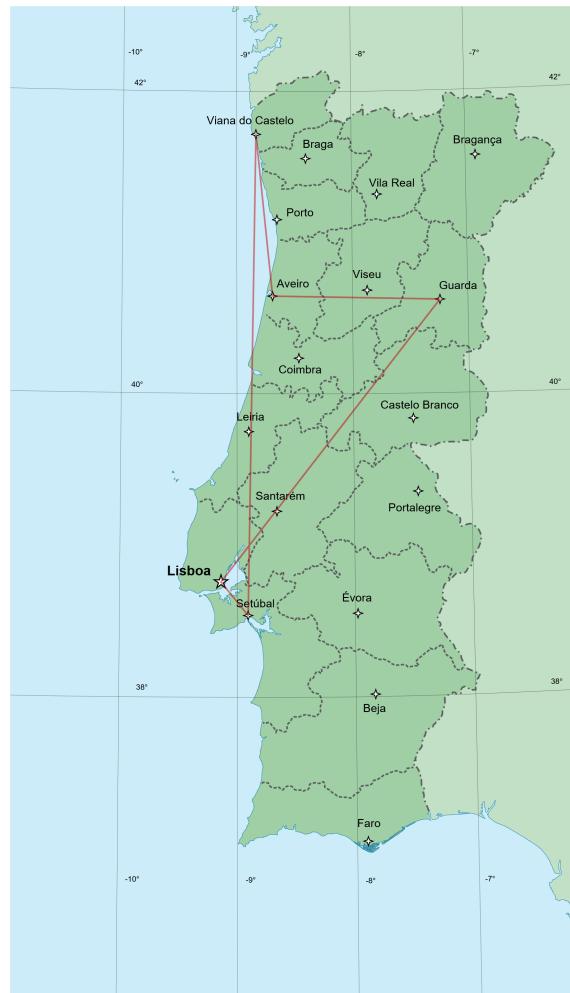


Análise de Resultados | Mapas gerados

88808 | Melhor trajeto para n = 5
Special 0

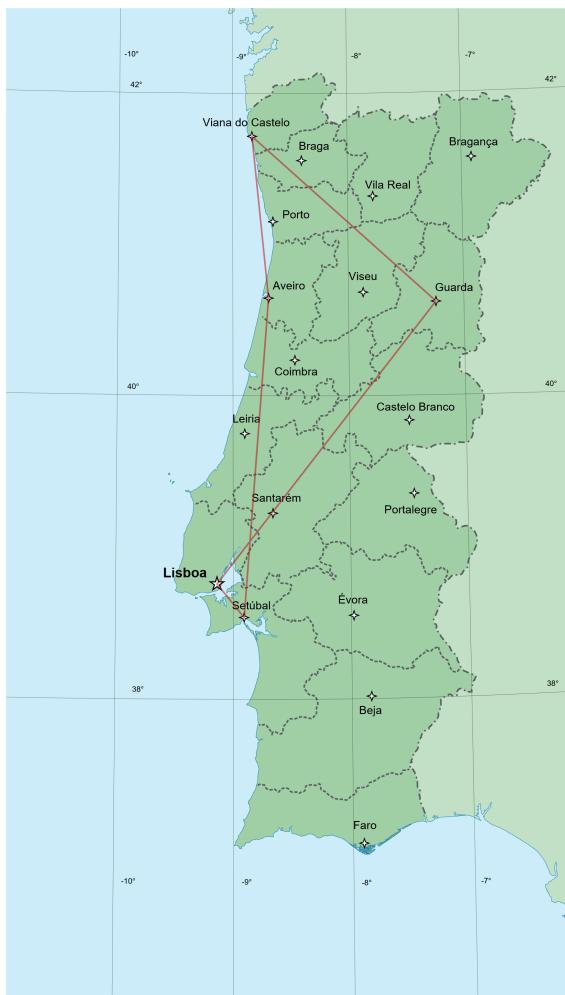


88808 | Melhor trajeto para n = 5
Special 1

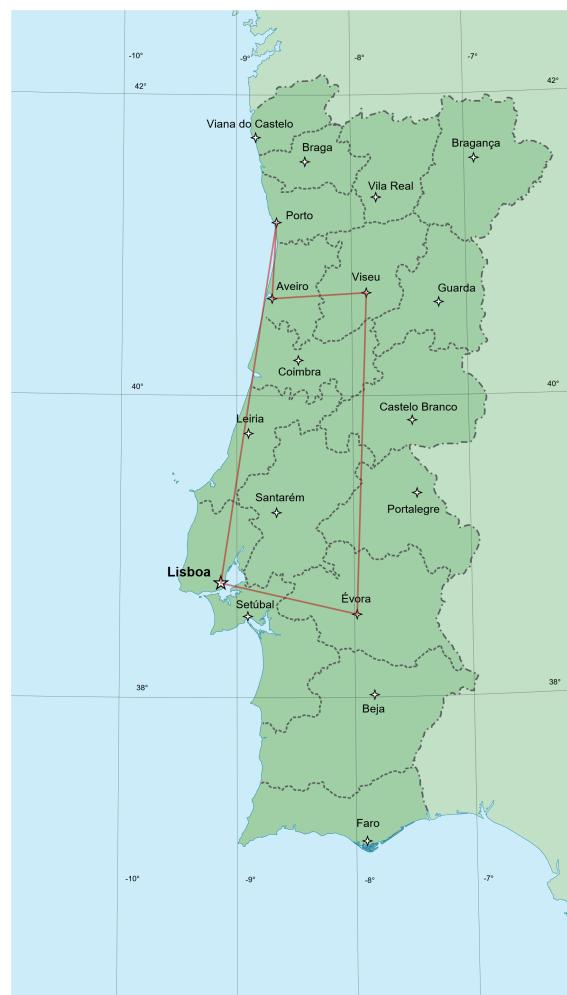


Análise de Resultados | Mapas gerados

88808 | Melhor trajeto para n = 5
Programação dinâmica



88886 | Melhor trajeto para n = 5
Programação dinâmica



Notas importantes e limitações

Notas importantes e limitações

Como foi dito anteriormente, os testes analisados anteriormente foram executados, para cada um dos números mecanográficos dos alunos, nos respetivos computadores pessoais. Ora, visto que os computadores possuem características diferentes, principalmente o processador, os tempos de execução demonstrados nas página 13 diferem ligeiramente. Apenas ligeiramente porque, visto que um dos processador é um INTEL CORE I5 de sétima geração e o outro é um INTEL CORE I7 de sexta geração, após uma comparação no site <https://ark.intel.com/>, concluímos que ambos os processadores são praticamente equivalentes, diferindo em poucos aspectos.

The screenshot shows the Intel Processor Comparison tool interface. At the top, there are navigation links for 'Produtos', 'Soluções', and 'Suporte', along with the Intel logo and language selection 'Brasil (Português)'. Below the header, there's a search bar and a link to 'Página inicial do suporte > Especificações do produto'. The main area features a 'Comparar produtos Intel®' section with a 'Realçar as diferenças' checkbox checked. It displays two products: 'Processador Intel® Core™ i5-7200U' and 'Processador Intel® Core™ i7-6700T'. Both products are shown with their respective logos and names.

O facto de ambos os computadores estarem equipados com um disco SSD foi preponderante na execução dos testes realizados, melhorando os tempos de execução, visto que um disco deste tipo aumenta a sua velocidade reduzindo os tempos de acesso à memória.

No entanto, o facto de existirem diversas aplicações a correm em paralelo com o nosso programa pode ter condicionado os tempos de execução do mesmo. Uma experiência realizada por nós foi o de correr o nosso programa em diferentes janelas do terminal ao mesmo tempo, prejudicando os tempos de execução, como mostrado na seguinte figura.

```
14) tsp_v1() finished in 229.279s (6227020800 tours generated)
   min 1481 [ 0, 1, 9, 7, 4,11, 8,12, 2, 3, 6,10,13, 5]
   max 5412 [ 0,11, 1, 8,13, 6, 9, 2, 4,10, 5, 3, 7,12]
```

Terminal 1 - 14 cidades
Tempo de execução 229.279 s

```
14) tsp_v1() finished in 239.312s (6227020800 tours generated)
   min 1481 [ 0, 1, 9, 7, 4,11, 8,12, 2, 3, 6,10,13, 5]
   max 5412 [ 0,11, 1, 8,13, 6, 9, 2, 4,10, 5, 3, 7,12]
```

Terminal 4 - 14 cidades
Tempo de execução 239.312 s

$$239.312 - 229.279 = 10,033 \text{ s de diferença}$$

Conclusões

Conclusões

Após a realização deste trabalho prático, concluímos que os objetivos propostos foram alcançados com sucesso. Conseguimos implementar os três métodos de resolução do Problema do Caixeiro Viajante (Força bruta, Permutações aleatórias e Programanção dinâmica), obtendo resultados satisfatórios e de acordo com a previsão, como já apresentados anteriormente. Com este trabalho, ambos os alunos do grupo fortaleceram os seus conhecimentos em diversos conceitos de programação abordados na Unidade Curricular de Algoritmos e Estruturas de Dados. É de salientar ainda que o trabalho de equipa e a superação de dificuldades foram fatores importantíssimos no sucesso do trabalho, melhorando as competências interpessoais de ambos os elementos do grupo.

Referências

https://en.wikipedia.org/wiki/Travelling_salesman_problem

<https://www.geeksforgeeks.org/travelling-salesman-problem-set-1/>

Slides da Unidade Curricular de Algoritmos e Estruturas de Dados da Universidade de Aveiro

Anexo

O anexo seguinte apresenta o código produzido pelos alunos para a implementação dos três métodos de resolução do Problema do Caixeiro Viajante descritos anteriormente, em linguagem de programação C. Note-se que este código foi produzido tendo por base o código já disponibilizado a todos os alunos da unidade curricular, que poderá ter sido alterado em algumas partes.

Código do ficheiro tsp.c:

```
//  
// Students  
// 88808 - João Miguel Nunes de Medeiros e Vasconcelos  
// 88886 - Tiago Carvalho Mendes  
//  
// AED, 2018/2019  
//  
// solution of the traveling salesman problem  
//  
  
#include <assert.h>  
#include <math.h>  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <sys/time.h>  
#include <sys/resource.h>  
#include <limits.h>  
  
#include "cities.h"  
#include "elapsed_time.h"  
  
//  
// record best solutions for tsp_v1  
//  
  
static int min_length,max_length,lc,histogram;  
static int min_tour[max_n_cities + 1],max_tour[max_n_cities + 1];  
static long n_tours;  
static int hist[10000];  
  
// tsp_v2 extra variables  
static int visited_all;  
static int best_distances[(1<<max_n_cities)][max_n_cities];  
static int current_city;  
static int control;  
static int best_position[(1<<max_n_cities)][max_n_cities];
```

Anexo | tsp.c

```
//  
// first solution (brute force, distance computed at the end, compute best and worst tours)  
//  
  
int computeTourLength(int n, int *a)  
{ // function to compute tour length in the tsp_v1  
    int i, tourLength = 0;  
  
    for(i = 0;i < n-1;i++)  
    {  
        tourLength += cities[a[i]].distance[a[i+1]];  
    }  
    tourLength += cities[a[n-1]].distance[a[0]];  
  
    // histogram of the length of all tours for 12 and 15 cities  
    if(histogram == 1 && (n == 12 || n == 15))  
    {  
        hist[tourLength]++;  
    }  
    return tourLength;  
}  
  
void updateLengths(int n, int tourLength, int *a)  
{ // function to update distances and tours (best and worst) in the tsp_v1  
    int i;  
  
    if(tourLength < min_length)  
    { // update min_length and min_tour  
        min_length = tourLength;  
        for(i = 0;i < n;i++)  
        {  
            min_tour[i] = a[i];  
        }  
    }  
    else if(tourLength > max_length)  
    { // update max_length and max_tour  
        max_length = tourLength;  
        for(i = 0;i < n;i++)  
        {  
            max_tour[i] = a[i];  
        }  
    }  
}
```

Anexo | tsp.c

```
void tsp_v1(int n,int m,int *a)
{
    int i,t,tourLength;

    if(m < n - 1)
        for(i = m;i < n;i++)
    {
        t = a[m];
        a[m] = a[i];
        a[i] = t;
        tsp_v1(n,m + 1,a);
        t = a[m];
        a[m] = a[i];
        a[i] = t;
    }
    else
    { // visit permutation
        n_tours++;
        tourLength = computeTourLength(n,a); // compute tour length
        updateLengths(n,tourLength,a); // update min_length, min_tour, max_length and max_tour
    }
}

void rand_perm(int n,int a[])
{ // function to generate random permutations in the tsp_v1
    int i,j,k,tourLength;

    for(i = 0;i < n;i++)
    {
        a[i] = i;
    }
    for(i = n - 1;i > 0;i--)
    {
        j = (int)floor((double)(i + 1) * (double)rand() / (1.0 + (double)RAND_MAX)); // range 0..i
        assert(j >= 0 && j <= i);
        k = a[i];
        a[i] = a[j];
        a[j] = k;
    }
    n_tours++;
    tourLength = computeTourLength(n,a);
    updateLengths(n,tourLength,a); // update min_length, min_tour, max_length and max_tour
}
```

Anexo | tsp.c

```
int tsp_v2(int mask, int position)
{
    if(mask == visited_all)
    {
        return cities[position].distance[0];
    }

    if(best_distances[mask][position] != 0)
    {
        return best_distances[mask][position];
    }

    int min = 100000000, max = 0, city;
    int best_city = 0;

    for(city = 0; city < current_city; city++)
    {
        if((mask & (1 << city)) == 0)
        {
            int d = cities[position].distance[city] + tsp_v2(mask | (1 << city), city);

            if(control == 0)
            {
                if(d < min)
                {
                    min = d;
                    best_city = city;
                }
            } else
            {
                if(d > max)
                {
                    max = d;
                    best_city = city;
                }
            }
        }
    }

    best_position[mask][position] = best_city;
    if(control == 0)
    {
        best_distances[mask][position] = min;
    }
    else
    {
        best_distances[mask][position] = max;
    }

    return best_distances[mask][position];
}
```

Anexo | tsp.c

```
//  
// main program  
//  
  
int main(int argc,char **argv)  
{  
    int n_mec,special,random,print,n,i,a[max_n_cities],tsp_v;  
    char file_name[64], file_name2[64];  
    double dt1;  
    FILE *file, *file2;  
  
    n_mec = 88886; // change later to n_mec = 88808  
    special = 0; // if you want asymmetric distances, change this to special = 1  
    random = 0; // if you want random permutations, change this to random = 1  
    histogram = 0; // if you want to make an histogram of the length of all tours, change this to histogram = 1  
    print = 0; // if you want to save the data to a .csv file, change this to 1  
    tsp_v = 2; // if you want to solve TSP problem using Dynamic Programming, change this to 2  
  
    init_cities_data(n_mec,special);  
    printf("data for init_cities_data(%d,%d)\n",n_mec,special);  
    fflush(stdout);  
  
    if(print != 0)  
    { // open file and initialize it  
        sprintf(file_name,"./Data/Special_%d/%d/%s.csv",special,n_mec,(random == 0) ? "tsp_data" : "tsp_random_data");  
        file = fopen(file_name,"w");  
        fprintf(file,"%s;%s;%s;%s;%s;%s\n","Number of cities (n)","Execution time (s)","minLength","minPath","maxLength","maxPath");  
    }  
  
    #if 1  
    print_distances();  
    #endif  
    for(n = 3;n <= max_n_cities;n++)  
    {  
        //  
        // try tsp_v1  
        //  
        dt1 = -1.0;  
        if(n <= 18)  
        {  
            (void)elapsed_time();  
            for(i = 0;i < n;i++)  
                a[i] = i;  
            min_length = 1000000000;  
            max_length = 0;  
            n_tours = 0;  
        }
```

Continua na página seguinte

Anexo | tsp.c

```
// choose permutations type
if(random == 0)
{
    if(tsp_v == 1)
    {
        tsp_v1(n,1,a); // no need to change the starting city, as we are making a tour
    }
    else
    { // tsp_v2 - dynamic programming approach
        current_city = n;
        for(int c = 0;c < 2;c++)
        {
            visited_all = (1<<current_city) - 1;
            for(int mask = 0;mask < (1<<n_cities);mask++)
                for(int position = 0;position < max_n_cities;position++)
                {
                    best_distances[mask][position] = 0;
                    best_position[mask][position] = 0;
                }
            // update min_tour or max_tour
            if(control == 0)
            {
                min_length = tsp_v2(1,0);
                int pos = 0; int indice = 0;
                for(int x1 = 1;x1 != visited_all;)
                {
                    int x2 = best_position[x1][pos];
                    min_tour[indice] = x2;
                    x1 = x1|(1<<x2);
                    pos=x2;
                    indice++;
                }
                for(int o = indice; o > 0; o--)
                {
                    min_tour[o] = min_tour[o-1];
                }
                min_tour[0] = 0;
                control = 1;
            }
            else
            {
                max_length = tsp_v2(1,0);
                int pos = 0; int indice = 0;
                for(int x1 = 1;x1 != visited_all;)
                {
                    int x2 = best_position[x1][pos];
                    max_tour[indice] = x2;
                    x1 = x1|(1<<x2);
                    pos=x2;
                    indice++;
                }
                for(int o = indice; o > 0; o--)
                {
                    max_tour[o] = max_tour[o-1];
                }
                max_tour[0] = 0;
                control = 0;
            }
        }
    }
}
else
{ // random permutations
    for(i = 0;i < 20000000;i++)
    {
        rand_perm(n,a);
    }
}
```

Continua na página seguinte

Anexo | tsp.c

```
dt1 = elapsed_time();
printf("%d) tsp_v1() finished in %8.3fs (%ld tours generated)\n",n,dt1,n_tours);

// print min_tour by index
printf(" min %5d [%min_length];
for(i = 0;i < n;i++)
{
    printf("%2d%s",min_tour[i],(i == n - 1) ? "]\n" : ",");
}

// print max_tour by index
printf(" max %5d [%max_length];
for(i = 0;i < n;i++)
{
    printf("%2d%s",max_tour[i],(i == n - 1) ? "]\n\n" : ",");
}

// save the computed data into a file
if(print != 0)
{
    fprintf(file,"%d;%8.3f;%d;[%n,dt1,min_length);
    for(i = 0; i < n;i++)
    {
        fprintf(file,"%d%s",min_tour[i],(i == n - 1) ? "];" : ",");
    }
    fprintf(file,"%d;[%max_length);
    for(i = 0; i < n;i++)
    {
        fprintf(file,"%d%s",max_tour[i],(i == n - 1) ? "]\n" : ",");
    }
}

if(histogram == 1 && (n == 12 || n == 15))
{
    sprintf(file_name2,"./Data/Special_%d/%d/%d_cities_tours_histogram.csv",special,n_mec,n);
    file2 = fopen(file_name2,"w");
    //fprintf(file2,"%s;%s\n","Tour","Occurrences");
    for(i = 0; i < 10000; i++)
    {
        fprintf(file2,"%d,%d\n",i,hist[i]);
    }
    fclose(file2);
}

fflush(stdout);
if(argc == 2 && strcmp(argv[1] "-f") == 0)
{
    min_tour[n] = -1;
    sprintf(file_name,"./Maps/Special_%d/%d/%s/min_%02d.svg",special,n_mec,(random == 0) ? "Normal" : "Random",n);
    make_map(file_name,min_tour);
    max_tour[n] = -1;
    sprintf(file_name,"./Maps/Special_%d/%d/%s/max_%02d.svg",special,n_mec,(random == 0) ? "Normal" : "Random",n);
    make_map(file_name,max_tour);
}
}

if(print == 1) {
    fclose(file);
}
return 0;
}
```

Fim do ficheiro tsp.c