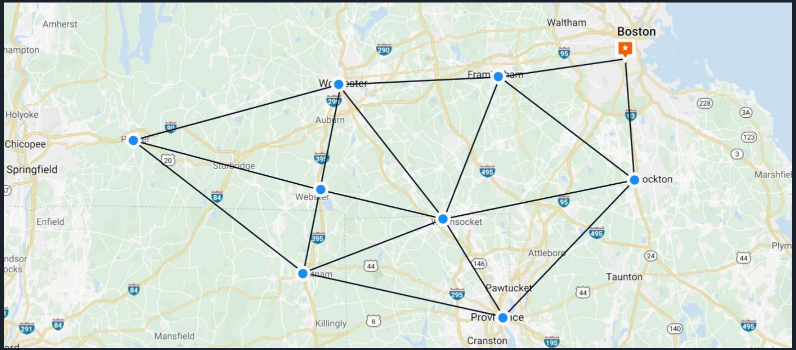
Travelling Salesman Problem

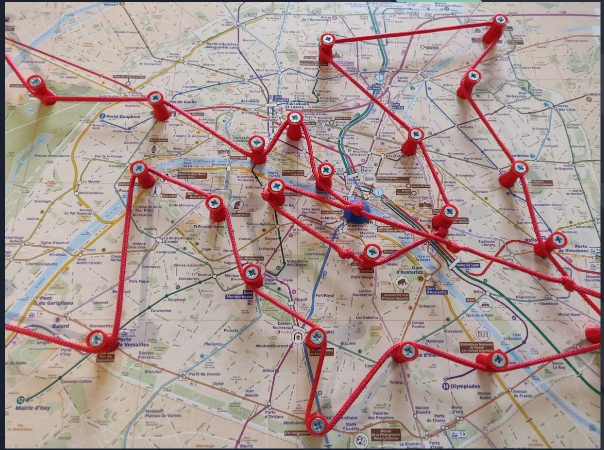
Desenho de Algoritmos 2022/2023

Francisco da Ana, up202108762
Francisco Lopes, up202108796
Maria Carlota Leita, up202005428

Problema

Dado um conjunto de cidades e as distâncias entre cada par de cidades, o objetivo é encontrar o menor caminho possível que visite cada cidade exatamente uma vez e retorne à cidade de partida.





Backtracking Algorithm
Explorar todas as combinações e encontrar a melhor

Inicialização: Começar com um caminho vazio e definir a cidade inicial como a cidade atual.

Gerar Permutações: Gerar todas as permutações possíveis das cidades não visitadas

Verificar Viabilidade: Verificar se adicionar a próxima cidade é viável (não visitada)

Atualizar Caminho: Adicionar a cidade viável, marcar como visitada, atualizar o comprimento do caminho

Backtracking: Se não for viável, remover a última cidade, fazer backtracking para a cidade anterior

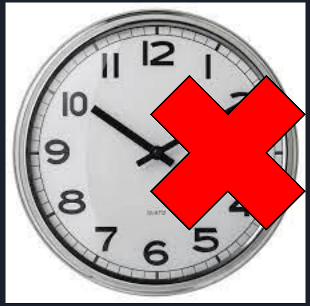
Terminação: Repetir até que todas as cidades sejam visitadas

Atualizar Caminho Mais Curto: Compare o caminho atual com o caminho mais curto encontrado

Saída: Retorne o caminho mais curto como a solução ótima

Complexidade:
O((V-1)!)

Resultados:



Extremamente ineficiente para grafos maiores

		Backtracking	
	N	distance	run time
shipping	14	86,7	1067
stadiums	11	341	62035
tourism	5	2600	0

Triangular Approximation Heuristic

Árvore de Extensão Mínima (MST): Construir uma Árvore de Extensão Mínima do grafo dado - usando o algoritmo de Prim. A MST é uma árvore que conecta todos os nós com o menor peso total das arestas.

Busca em Profundidade (DFS): Realizar uma travessia de Busca em Profundidade começando no nó 0 na MST. Essa travessia explora a árvore e visita cada nó exatamente uma vez.

Durante a travessia DFS, visitar os nós em pre-order Isto significa visitar o nó atual, em seguida, visitar o seu vizinho não visitado de menor número. Por fim, visitar o vizinho não visitado de menor número desse vizinho. Repetir este processo até que todos os nós da MST sejam visitados.

Fechar o Ciclo: Uma vez que todos os nós forem visitados, retornar ao nó inicial.

Solução Aproximada: A ordem na qual os nós foram visitados durante a travessia DFS representa uma solução aproximada para o TSP. O caminho formado pelos nós visitados, incluindo o retorno ao nó inicial, representa o percurso aproximado.

```
pair<double, list<unsigned int>> Graph::TSP_TriangularApproximation(){
    prim_generate_MST();
    list<unsigned int> order;
    dfsMST(0, order);
    order.push_back(0);
    double cost = getPathCost(order);
    return make_pair(cost, order);
}
```

Triangular Approximation Heuristic
Reflexão

Complexidade

Algoritmo de Prim: O((V+E)* logV)

DFS: O(V + E)

Total: O((V+E)* logV)

Resultado:

Algoritmo escalável ao ponto de produzir resultados para todos os grafos do nosso dataset (à exceção do shipping.csv, que não é completo). Contudo os resultados obtidos não são os melhores no que toca à minimização da distância total (fator aproximação)

			Triangular Approximation H.		
		Nodes	distance 1	runtime	(N + E)*log(N)
Toy Graphs	Shipping	14	not possible	-	120,3434437
	Stadiums	11	398,1	0	68,73191722
	Tourism	5	2600	0	10,48455007
Real World Graphs	graph1	1000	115000	2125	1501500
	graph2	5000	3980000	27007	46246372,48
	graph3	10000	6940000	87399	200020000
Extra Fully-Connected Graphs	edges_25	25	364840	1	454,3305028
	edges_50	50	574830	2	2166,186756
	edges_75	75	642706	6	5343,924601
	edges_100	100	722729	14	10100
	edges_200	200	904553	59	46250,70291
	edges_300	300	128000	140	111842,0247
	edges_400	400	1370000	245	208685,2113
	edges_500	500	1520000	451	338045,993
	edges_600	600	1650000	680	500900,6704
	edges_700	700	194000	980	698044,8041
	edges_800	800	2010000	1136	930150,0318
	edges_900	900	2140000	1767	1197797,625

Cheapest Insertion Heuristic

1. Inicialmente, o algoritmo define todos os nós como não visitados e cria um caminho vazio.
2. O nó 0 é adicionado ao caminho inicial e marcado como visitado.
3. Um nó r é selecionado para ser o próximo nó a ser adicionado ao caminho. Inicialmente, o nó mais próximo de 0 é escolhido.
4. Enquanto o tamanho do caminho for menor que o número total de nós:
5. a. Para cada nó k não visitado:
6. i. É calculado o custo da inserção de k entre cada par de nós já presentes no caminho. Isso é feito comparando as distâncias entre os nós e encontrando a inserção que resulta no menor custo adicional.
7. ii. O custo da inserção mais barata é armazenado e o índice de inserção correspondente é registrado.
8. b. O nó r é inserido no caminho na posição determinada pelo índice de inserção mais barata.
9. c. O nó r é marcado como visitado.
10. Ao final do loop, o nó 0 é adicionado novamente ao final do caminho para completar o ciclo.
11. O custo total do caminho é calculado.
12. O algoritmo retorna o custo e o caminho resultante.

PORQUÊ?

Boa capacidade de explorar localmente as melhores inserções entre nós já presentes no caminho.

Aproveita um caminho inicial próximo de uma solução razoável e busca inserções baratas em todas as etapas.

Cheapest Insertion Heuristic
Reflexão

Extra Fully-Connect ed Graphs	edges_25	296563	0
	edges_50	453786	6
	edges_75	565186	27
	edges_100	582769	78
	edges_200	756709	1139
	edges_300	1010000	5733
	edges_400	1200000	18312
	edges_500	1230000	45094
	edges_600	1380000	97771
	edges_700	1530000	202312
	edges_800	1660000	413809
	edges_900	1770000	1076570

Complexidade:

Loop principal: O(n)

Cálculo da melhor posição para inserir o novo node: O(n^2)

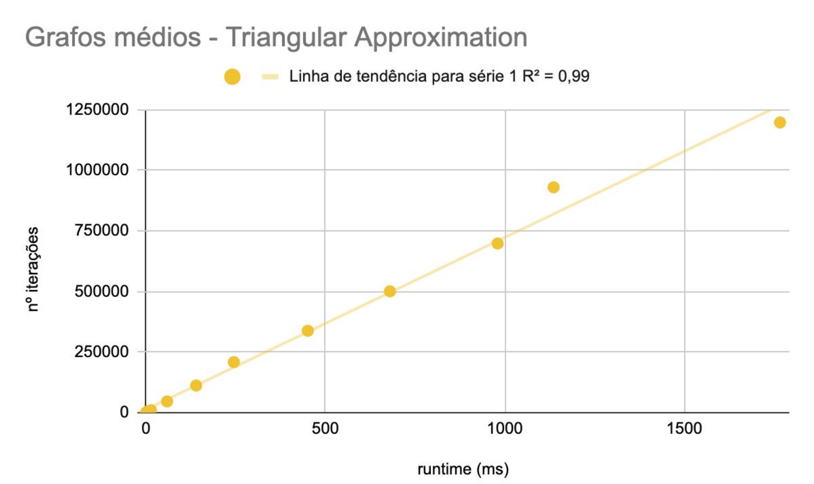
Total: O(n^3)

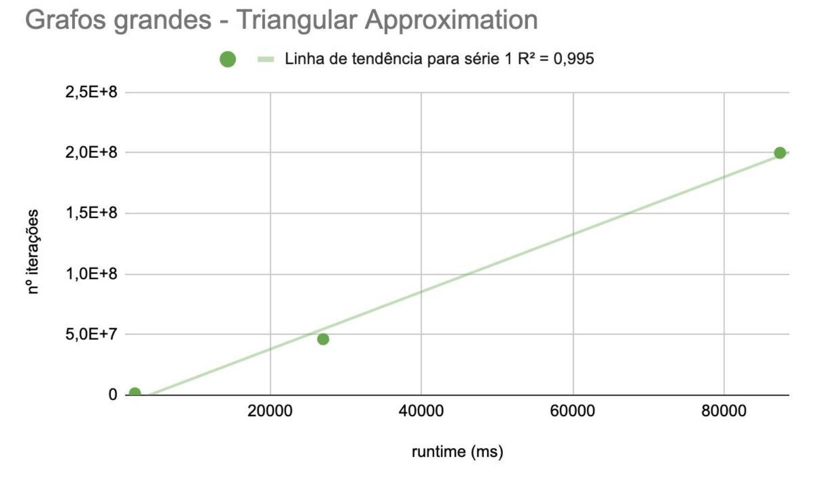
Resultado:

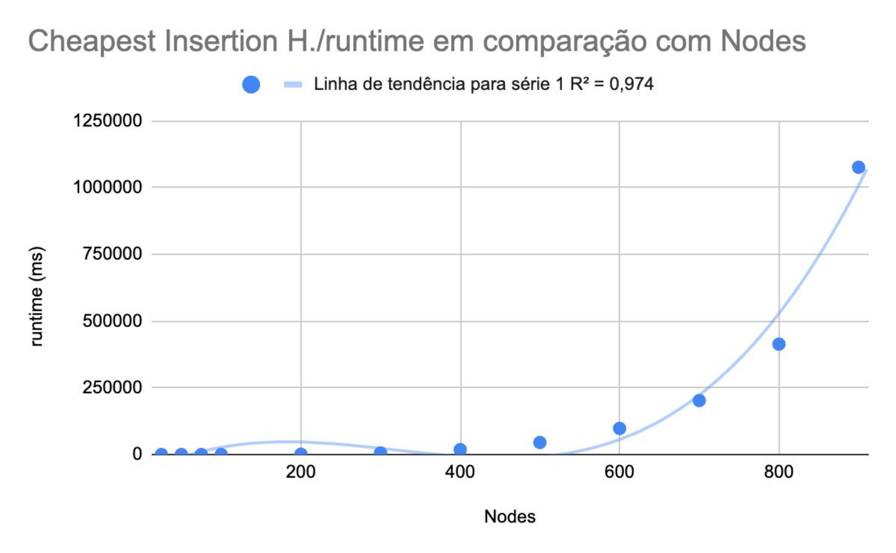
Valores bastante otimizados. Contudo, a complexidade faz-se sentir um pouco à medida que o tamanho do grafo aumenta. Em geral, passa por perder um pouco mais de tempo para obter melhores respostas.

Resultados obtidos pelos diferentes algoritmos

				Backtracking			Triangular Approximation H.			Cheapest Insertion H.		
		Nodes	Edges	distance	runtime	N!	distance 1	runtime	(N + E)*log(N)	distance 2	runtime	N^3
Toy Graphs	Shipping	14	91	86,7	1067	87178291200	not possible	-	120,3434437	not possible	-	2744
	Stadiums	11	55	341	62035	39916800	398,1	0	68,73191722	348,6	0	1331
	Tourism	5	10	2600	0	120	2600	0	10,48455007	2600	0	125
Real World Graphs	graph1	1000	499500	-	-		115000	2125	1501500	-	-	1000000000
	graph2	5000	12497500	-	-		3980000	27007	46246372,48	-	-	125000000000
	graph3	10000	49995000	-	-		6940000	87399	200020000	-	-	1000000000000
Extra Fully-Connected Graphs	edges_25	25	300	-	-		364840	1	454,3305028	296563	0	15625
	edges_50	50	1225	-	-		574830	2	2166,186756	453786	6	125000
	edges_75	75	2775	-	-		642706	6	5343,924601	565186	27	421875
	edges_100	100	4950	-	-		722729	14	10100	582769	78	1000000
	edges_200	200	19900	-	-		904553	59	46250,70291	756709	1139	8000000
	edges_300	300	44850	-	-		128000	140	111842,0247	1010000	5733	27000000
	edges_400	400	79800	-	-		1370000	245	208685,2113	1200000	18312	64000000
	edges_500	500	124750	-	-		1520000	451	338045,993	1230000	45094	125000000
	edges_600	600	179700	-	-		1650000	680	500900,6704	1380000	97771	216000000
	edges_700	700	244650	-	-		194000	980	698044,8041	1530000	202312	343000000
	edges_800	800	319600	-	-		2010000	1136	930150,0318	1660000	413809	512000000
	edges_900	900	404550	-	-		2140000	1767	1197797,625	1770000	1076570	729000000


Grafos médios - Triangular Approximation
Linha de tendência para série 1 R² = 0,99
1250000
1000000
750000
500000
250000
0
nº iterações
0
500
1000
1500
runtime (ms)


Grafos grandes - Triangular Approximation
Linha de tendência para série 1 R² = 0,995
2,5E+8
2,0E+8
1,5E+8
1,0E+8
5,0E+7
0
nº iterações
20000
40000
60000
80000
runtime (ms)


Cheapest Insertion H./runtime em comparação com Nodes
Linha de tendência para série 1 R² = 0,974
1250000
1000000
750000
500000
250000
0
runtime (ms)
200
400
600
800
Nodes

Resultados Triangular Approximation (distance 1) VS Resultados Cheapest Insertion(distance 2)

		Comparison
		distance 2 / distance 1
Extra Fully-Connected Graphs	edges_25	81,29%
	edges_50	78,94%
	edges_75	87,94%
	edges_100	80,63%
	edges_200	83,66%
	edges_300	789,06%
	edges_400	87,59%
	edges_500	80,92%
	edges_600	83,64%
	edges_700	788,66%
	edges_800	82,59%
	edges_900	82,71%
		83,17%

Caminhos encontrados pela Cheapest Insertion Heuristics têm em média 83,17% da distância dos caminhos encontrados pela Triangular Approximation

Conclusões

- Backtracking, de facto, inutilizável para um contexto real
- Triangular Approximation é uma alternativa viável com uma complexidade temporal muito inferior.
- Cheapest Insertion com ótimos resultados mas não tão eficiente. Dispensa de algum tempo para melhores resultados.
- Tínhamos implementado, antes, a Nearest Insertion Heuristic. Esta não procura a inserção numa posição tão ótima, pelo que as respostas não eram tão boas. A sua complexidade era O(n^2) mas optámos por esta implementação.