Travelling Salesman Problem

Work done by:

Adriano Machado
José Evans
Rodrigo Moisés

Problem

Finding the shortest route for a salesman to visit a set of cities and return to the starting city, while visiting each city exactly once.

Backtracking Algorithm - A systematic approach to explore all possible permutations and find the optimal solution

Initialization: Start with an empty path and set the initial city as current

Generate Permutations: Generate all possible permutations of unvisited cities

Check Feasibility: Ensure adding the next city is feasible (unvisited)

Update Path: Add feasible city, mark as visited, update path length

Backtrack: If not feasible, remove last city, backtrack to previous

Termination: Repeat until all cities visited

Update Shortest Path: Compare current path with shortest path found

Output: Return shortest path as the optimal solution

Backtracking Algorithm - Conclusions

- The algorithm could only be applied to the Toy Graphs, since its complexity is exponencial **O((V-1)!)** , and the time it took too long to solve the problem in bigger graphs.
- It explores all possible permutations and finds the optimal route, but it is extremely inefficient. If we compare with the results obtained with the triangular approximation heuristic or with the nearest insertion, we quickly realize that it is not an efficient solution.

Toy Graphs	Shipping	86.7	598
	Stadiums	341	30539
	Tourism	2600	1

Triangular Approximation Heuristic

Minimum Spanning Tree (MST): Construct a Minimum Spanning Tree of the given graph using Prim's algorithm. The MST is a tree that connects all the nodes with the minimum total edge weight.

Depth-First Search (DFS): Perform a Depth-First Search traversal starting from node 0 in the MST. This traversal explores the tree and visits each node exactly once.

Triangular Pattern: During the DFS traversal, visit the nodes in a triangular pattern. This means visiting the current node, then visiting its lowest numbered unvisited neighbor, and finally visiting the lowest numbered unvisited neighbor of the neighbor. Repeat this process until all nodes in the MST are visited.

Complete the Cycle: Once all nodes are visited, return to the starting node to complete the cycle.

Approximate Solution: The order in which the nodes were visited during the DFS traversal represents an approximate solution to the TSP. The path formed by the visited nodes, including the return to the starting node, represents the approximate tour.

```
vector<Node*> Graph::tspTriangular(double* distance) {
    primMST();

    vector<Node*> H = dfsTriangular( node: nodes[0]);
    H.push_back(nodes[0]);

    *distance = 0;
    for (int i = 0; i < H.size() - 1; i++) {
        Node* source = H[i];
        Node* dest = H[i + 1];
        *distance += source->getDistanceTo( node: dest);
    }
    return H;
}
```

Triangular Approximation Heuristic

- The time complexity of the algorithm depends mainly on the MST construction and DFS. For the Prim's algorithm the time complexity is O(|E| log|V|). For DFS it is O(|V|). So we can simplify the complexity of this heuristic to O(|E| log|V|).

- We could use this algorithm for every graph except the first one, since it's a very efficient algorithm, although the results it produces are not the best ones, since it is an approximation algorithm.

	Triangular Aproximation	
	distance	runtime
Shipping	Not possilbe	-
Stadiums	398.1	0
Tourism	2600	0
graph1	$1.14154 * 10^6$	1691
graph2	$3.3933 * 10^6$	17955
graph3	$5.89534 * 10^6$	56487
edges_25	349573	3
edges_50	554134	15
edges_75	627035	26
edges_100	681458	41
edges_200	909414	61
edges_300	$1.19689 * 10^6$	142
edges_400	$1.34421 * 10^6$	229
edges_500	$1.49618 * 10^6$	370
edges_600	$1.61821 * 10^6$	524
edges_700	$1.75767 * 10^6$	817
edges_800	$1.8649 * 10^6$	1025
edges_900	$2.05297 * 10^6$	1326

Cheapest Insertion Heuristic

It finds the city not already in the tour that when placed between two connected cities in the subtour will result in the shortest possible tour. It inserts the city between the two connected cities, and repeats until there are no more insertions left.

1. Start with an initial tour that includes the node 0 and its closest adjacent node.
2. Select a city that is not yet part of the tour. This city will be inserted into the existing tour.
3. Compute the cost of inserting the selected city between each pair of cities in the current tour. The cost is determined by calculating the additional distance that would be traveled if the selected city were inserted at that position.
4. Choose the pair of cities where inserting the selected city results in the smallest increase in total distance. This is referred to as the "cheapest" insertion.
5. Insert the selected city into the chosen position, modifying the tour.
6. Repeat steps 2-5 until all cities have been inserted into the tour.
7. Finally, return to the starting city to complete the tour.
8. Calculate the distance between the nodes in the tour and return it.

Cheapest Insertion Heuristic

- The main loop involves inserting each remaining city into the current tour. For each city, the algorithm computes the cost of insertion for each possible position in the tour. This step requires iterating over the tour, which takes O(n) time. Since there are n cities, this step takes O(n^2) time.

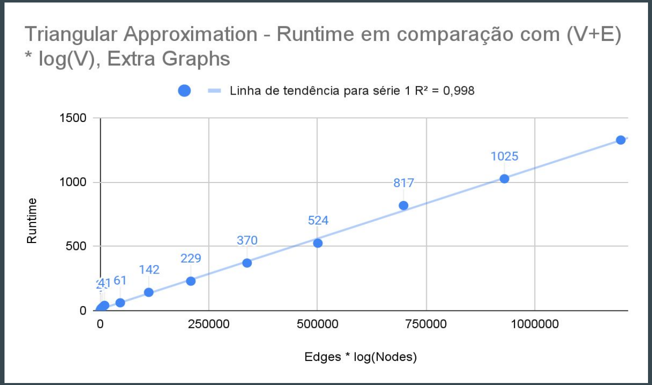
  Since there are n cities, the main loop will repeat n times, resulting in an additional factor of O(n).

  The expected complexity will be O(n^3).

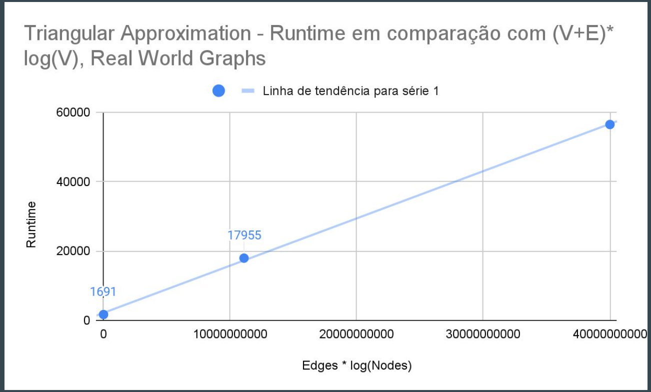
Cheapest Insertion Heuristic	
distance 2	runtime
Not possible	-
348.6	0
2600	0
-	-
-	-
-	-
296563	5
453786	16
565186	63
582769	133
756709	1550
1,00682E+06	7959
1,20369E+06	25976
1,22740E+06	83076
1,37716E+06	245956
1,53032E+06	545927
1,65924E+06	996020
1,77132E+06	1727710

Resultados obtidos pelos diferentes algoritmos

	Backtracking		Triangular Approximation		Cheapest Insertion Heuristic		Comparison
	distance	runtime	distance 1	runtime	distance 2	runtime	**distance 1 / distance 2 - 1**
Shipping	86.7	598	Not possilbe	-	Not possible	-	
Stadiums	341	30539	398.1	0	348.6	0	
Tourism	2600	1	2600	0	2600	0	
graph1	-	-	$1.14154*10^6$	1691	-	-	
graph2	-	-	$3.3933*10^6$	17955	-	-	
graph3	-	-	$5.89534*10^6$	56487	-	-	
edges_25	-	-	349573	3	296563	5	17,87%
edges_50	-	-	554134	15	453786	16	22,11%
edges_75	-	-	627035	26	565186	63	10,94%
edges_100	-	-	681458	41	582769	133	16,93%
edges_200	-	-	909414	61	756709	1550	20,18%
edges_300	-	-	1,19689E+06	142	1,00682E+06	7959	18,88%
edges_400	-	-	1,34421E+06	229	1,20369E+06	25976	11,67%
edges_500	-	-	1,49618E+06	370	1,22740E+06	83076	21,90%
edges_600	-	-	1,61821E+06	524	1,37716E+06	245956	17,50%
edges_700	-	-	1,75767E+06	817	1,53032E+06	545927	14,86%
edges_800	-	-	1,86490E+06	1025	1,65924E+06	996020	12,39%
edges_900	-	-	2,05297E+06	1326	1,77132E+06	1727710	15,90%
					Average		17,22%



Analysis of extra graphs time complexity



Analysis of real world graph time complexity

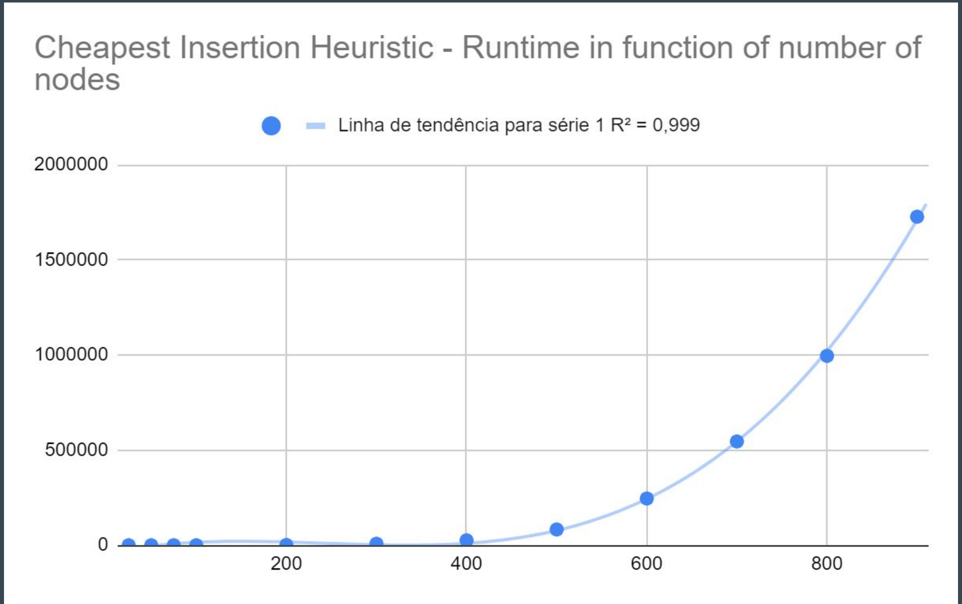
V	E	(V + E) * log(V)	Runtime
25	300	454,3305028	3
50	1225	2166,186756	15
75	2775	5343,924601	26
100	4950	10100	41
200	19900	46250,70291	61
300	44850	111842,0247	142
400	79800	208685,2113	229
500	124750	338045,993	370
600	179700	500900,6704	524
700	244650	698044,8041	817
800	319600	930150,0318	1025
900	404550	1197797,625	1326
1000	500000	1503000	1691
5000	3000000000	11096928508	17955
10000	1000000000(	40000040000	56487

Data table for Triangular Approximation

The data confirms the theoretical time complexity predictions, since runtime seems to be linear with (V+E) * log(V).

Temporal Complexity Analysis of Cheapest Insertion Heuristic

In theory, a complexity of O(V^3) was expected, which was verified since the data fit a 3rd degree polynomial regression, confirmed by $R^2 = 0.999$.



Distance Values Comparison for Triangular Approximation and Cheapest Heuristics, on the extra graphs

Triangular Approximation	Nearest Insertion Heuristics	Comparison
distance 1	distance 2	distance 1 / distance 2 - 1
349573	296563	17,87%
554134	453786	22,11%
627035	565186	10,94%
681458	582769	16,93%
909414	756709	20,18%
1,19689E+06	1,00682E+06	18,88%
1,34421E+06	1,20369E+06	11,67%
1,49618E+06	1,22740E+06	21,90%
1,61821E+06	1,37716E+06	17,50%
1,75767E+06	1,53032E+06	14,86%
1,86490E+06	1,65924E+06	12,39%
2,05297E+06	1,77132E+06	15,90%
	Average	17,22%

Distances obtained using the Cheapest Insertion Heuristic were, on average, 17.22% smaller than distances obtained using Triangular Approximation. However, as we have seen the time difference is huge. On the bigger graphs, using the insertion, it takes several minutes to output a result, while using the triangular produces almost instantaneous results.

Conclusions

- The backtracking algorithm was only usable on the toy graphs, since its complexity is exponential, but it worked and provided good shortest path solutions.

- The triangular approximation algorithm is an efficient approximation to solve this problem, useful for every graph. We can use it when we don’t need a high quality answer, but a fast response. The time values that we obtained were the expected theoretically.

Conclusions

- Firstly we tried to implement the nearest insertion algorithm, which is similar to the cheapest insertion, but with a time complexity O(n^2). However, the values were not great, and after implementing the cheapest insertion, we thought it would be more useful to have an algorithm that gave us good answers, even if it'd take longer. The cheapest algorithm produces decent results, but is indeed not efficient at all.

- The time complexity of the cheapest insertion was the theoretically expected.

- The distances were 17.2% smaller than the ones returned by the triangular approximation.