Travelling Salesman

Katherine S, Joshua W, Ethan L, Sahil S, Kirollos W

February 2024

1 Introduction

The Travelling Salesman Problem also known as TSP is a simple problem in concept. The problem is trying to solve the most efficient route to all destinations only once and returning to the starting point. There are three main ways to solve this problem: brute force, nearest neighbour, and dynamic programming. Brute force involves calculating all possible routes to find the shortest one. Nearest neighbour also known as the greedy method comprises of choosing the shortest option at the time. Dynamic programming is a way to solve the TSP using computers to solve for the most efficient route.

2 History

The history of the TSP can be traced back to the 1800s. In 1832, the Irish mathematician W.R. Hamilton described the Icosian Game, a puzzle involving the traversal of all edges of a dodecahedron exactly once. While not explicitly the Traveling Salesman Problem, the concept of visiting all vertices once sparked interest in similar problems.

The modern formulation of the TSP dates to the 1930s and 1940s. In 1930, Karl Menger, a mathematician from Austria, introduced the concept of the TSP in his study of the topology of networks. He explored the problem of finding the shortest closed loop through a given set of points.

The problem gained significant attention in the 1950s when Merrill Flood and Melvin Dresher of the RAND Corporation presented it as a mathematical challenge in the context of logistics and operations research. They were inspired by the problem of a traveling salesman seeking the most efficient route to visit a set of cities and return home.

In 1954, George Dantzig, Ray Fulkerson, and Selmer Johnson proposed the cutting-plane method to solve integer programming problems. While not directly solving the TSP, this laid the groundwork for future algorithms.

During the 1960s and 1970s, researchers developed various algorithms to tackle the TSP, including branch and bound methods and dynamic programming techniques. In 1972, Nicos Christofides proposed an algorithm known as the Christofides algorithm, which guarantees a solution within 3/2 times the optimal solution for any instance of the TSP.

Since then, the TSP has remained a central problem in the fields of optimization, theoretical computer science, and operations research. It has practical applications in logistics, transportation, manufacturing, and many other areas where efficient routing is essential. The ongoing quest for more efficient algorithms and techniques to solve the TSP continues to be an active area of research.

3 Algorithms

3.1 Brute Force

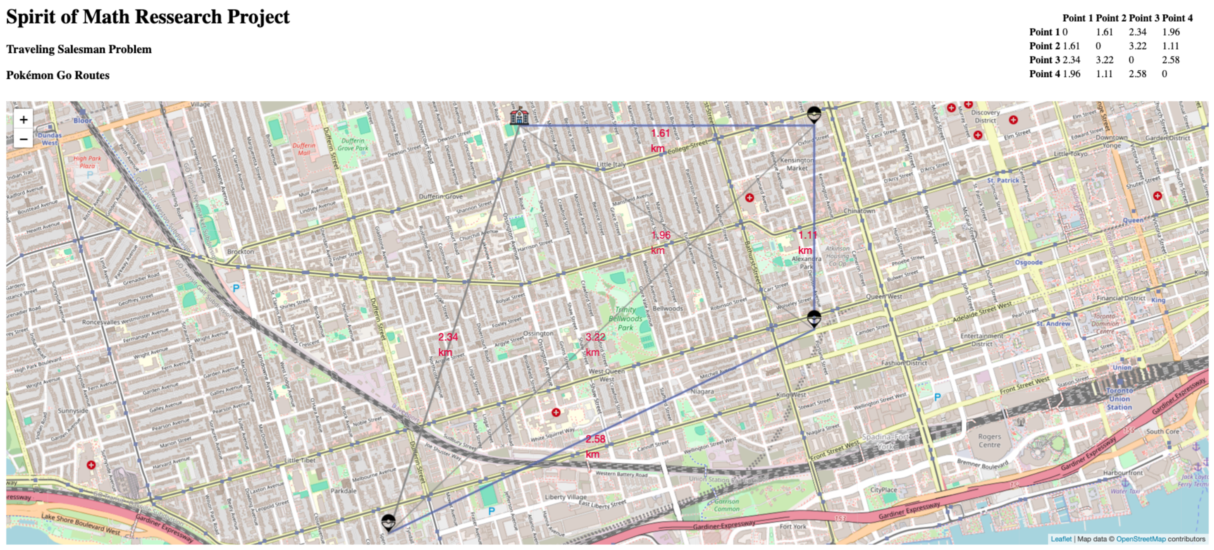
3.1.1. Background - Pokémon Go Game

In Pokémon Go, trainers can obtain essential items such as Poké Balls, potions, and other in-game resources by visiting designated locations called Poké Stops, where they can spin the stop's icon on their mobile device to receive a variety of items that aid in their Pokémon-catching adventures.

As a dedicated Pokémon Trainer, he would like to plan a journey to visit every Poké Stop near his school. The goal is to optimize the route for the shortest distance traveled while ensuring a return to his school.

3.1.2. Problem Statement - Simple version

As a Pokémon Trainer attending one of Toronto public high schools, he has identified three Poké Stops in his school's neighborhood, labeled as nodes 2, node 3, and node 4 (refer to Figure 2 below). Starting from his school (labeled as node 1), his objective is to find the shortest possible route that allows him to visit all three Poké Stops and return to his school.



Spirit of Math Ressearch Project

Traveling Salesman Problem

Pokémon Go Routes

	Point 1	**Point 2**	**Point 3**	**Point 4**
Point 1	0	1.61	2.34	1.96
Point 2	1.61	0	3.22	1.11
Point 3	2.34	3.22	0	2.58
Point 4	1.96	1.11	2.58	0

Figure 1: Pokémon Map to demonstrate TSP

For more information about the Pokémon map, visit replit.com. [7]

Figure 2 below is a simplified drawing representing the previous problem:

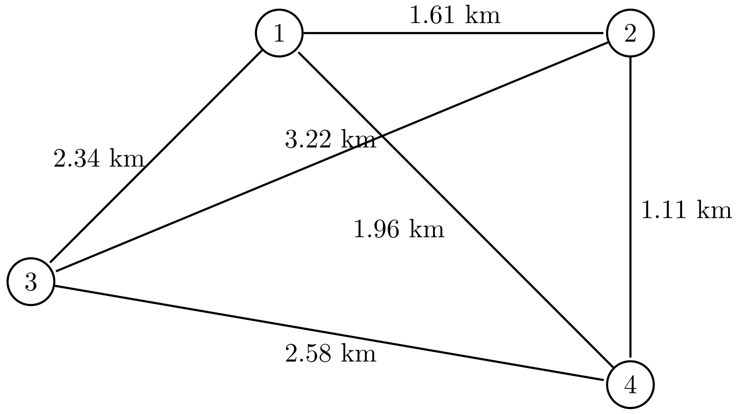


Figure 2: A Simplified Map

3.1.3. Brute Force Approach Analysis [6]

1. Represent nodes and paths in the matrix representation (based on Figure 2)

   A matrix representation of a graph of 4 nodes and 6 paths:

$$\begin{array}{c} \\ node1 \\ node2 \\ node3 \\ node4 \end{array} \begin{array}{c} \begin{array}{cccc} node1 & node2 & node3 & node4 \end{array} \\ \begin{pmatrix} 0 & 1.61 & 2.34 & 1.96 \\ 1.61 & 0 & 3.22 & 1.11 \\ 2.34 & 3.22 & 0 & 2.58 \\ 1.96 & 1.11 & 2.58 & 0 \end{pmatrix} \end{array}$$

   A total of 6 permutations (branches) in a tree representation:

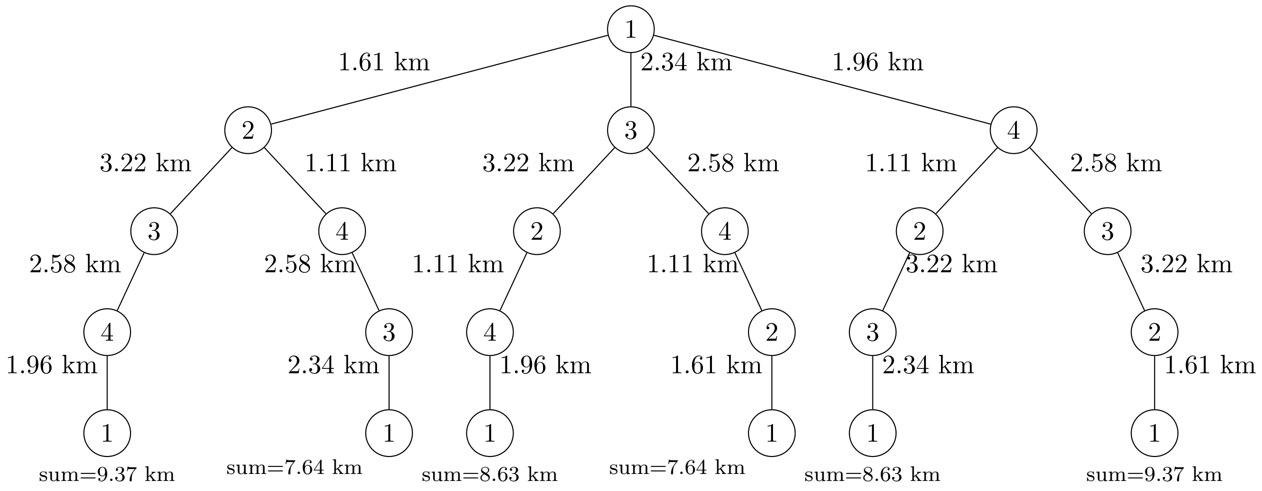


Figure 3: Brute Force Algorithm Analysis for Pokémon Go Routes

2. Calculate total distance for each permutation for every branch of the tree (as shown in Figure 3)

3. Find the minimum distance (e.g. node 1 - node 2 - node 4 - node 3 - node 1 = 7.64 km)

3.1.4. Computation Costs

If n presents the number of nodes, all possible permutations of node orderings is calculated as:

$$(n-1)!$$

For example, when $n = 4, 3! = 3 \times 2 \times 1 = 6$.

3.1.5. Pros and Cons of Brute Force

Pros

1. Simplicity: it is straightforward to implement, making them suitable for quick solutions and easy understanding.

2. Guaranteed Solution: it guarantees finding the optimal solution, as it exhaustively explores all possibilities.

Cons

1. Inefficiency: it is highly inefficient, especially for large input sizes, as they examine all possible solutions without exploiting problem-specific

2. Computational Complexity: The time and space complexity making it impractical. When $n = 11$, there is $3,628,800(10!)$ permutations.

3.2 Nearest Neighbor

3.2.1. Overview and Brief History

One of the most common and intuitive solutions is known as the "Nearest neighbour" solution. Starting at an arbitrary point, lines are drawn from each successive point to the one nearest to it, hence the name "nearest neighbour".

This method was specifically noted by Carl Menger, who studied the travelling salesman problem during the 1930s. The paper "*On the History of Combinatorial Optimization (till 1960)*", Karl Menger describes the difficulty of the brute-force method and also the lack of an efficient algorithm for a perfect solution,

"*Of course, this problem is solvable by finitely many trials. Rules which would push the number of trials below the number of permutations of the given points, are not known. The rule that one first should go from the starting point to the closest point, then to the point closest to this, etc., in general does not yield the shortest route.*"

[2]

3.2.2. Efficiency of Method

The nearest neighbour method will usually yield a route 25 percent longer than the ideal path for a number of points on a plane. However, specific arrangements might see this method finding a worse, or even the worst possible solution to the problem.

This algorithm will usually not find the optimal solution. However, it can and often is used to find a baseline for further improvement by other methods. The nearest neighbour method is notably a "greedy" method, seeking the best short-term outcome for each successive line drawn. Other optimization methods, will sometimes accept worse solutions temporarily to try to find better overall solutions.

One preliminary method to gauge the success of the nearest neighbour method is to compare the length of the first half of the tour to the last half. If the tours are roughly equal in length, then it is likely that the method has given a reasonable baseline. However, if the lengths are not close to equal, then it is likely that the algorithm was too 'greedy', sacrificing too much overall efficiency for a better route in the short term. [5]

Note that in the example shown previously, the ideal solution is discovered by the nearest neighbor method, beginning at any of the points, but in a slightly more complex diagram, the method begins to fall apart.

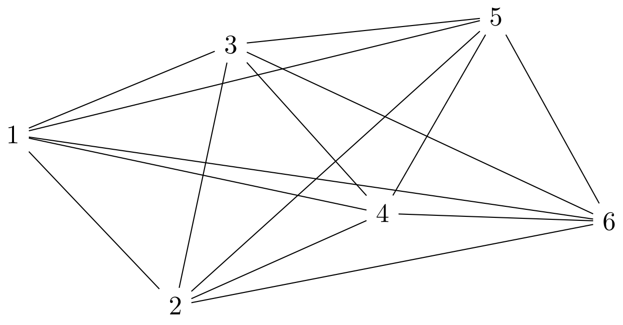
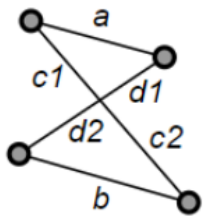


Figure 4: Nearest Neighbor Example

$$\begin{array}{c|cccccc} & node1 & node2 & node3 & node4 & node5 & node6 \\ \hline node1 & 0 & 2.21 & 2.34 & 3.65 & 4.18 & 4.78 \\ node2 & 2.21 & 0 & 2.67 & 2.12 & 4.06 & 3.99 \\ node3 & 2.34 & 2.67 & 0 & 2.08 & 2.39 & 3.91 \\ node4 & 3.65 & 2.12 & 2.08 & 0 & 2.78 & 0.97 \\ node5 & 4.18 & 4.06 & 2.39 & 2.78 & 0 & 3.01 \\ node6 & 4.78 & 3.99 & 3.91 & 0.97 & 3.01 & 0 \end{array}$$

Use the nearest neighbor method to find a solution, starting from node 3. What is your solution? Look at the length of the first half of the tour compared to the last and compare them. How greedy was the algorithm in this case? See if you can find a better one by starting at other nodes or with any other method.



$$a < c_1 + d_1$$

$$b < c_2 + d_2$$

$$a + b < (c_1 + c_2) + (d_1 + d_2)$$

Figure 5: Path swap explanation

3.2.3. Further Optimization

One simple method that can be used to immediately shorten a given path is the act of swapping lines that cross over each other. As the figure above shows, this will always yield a better path because of the triangle inequality.

Homework Problem Take a look at Dantzig 49, one of the first large-scale TSPs that was solved. Look at the map and the distance matrix. Start at a city of your choice and use the nearest neighbor method to find the next 20 cities. Now compare your solution so far to the ideal one found in 1954. Note that only 42 cities are shown out of the original 49, since the solution passed naturally through the 7 cities not listed. You can use this Google Earth project to plot the cities. https://earth.google.com/earth/d/1ElfLqJ7E7alScPKBDwD0J2hwYUdZXutu?usp=sharing

1. Manchester	2. Montpelier	3. Detroit
4. Cleveland	5. Charleston	6. Louisville
7. Indianapolis	8. Chicago	9. Milwaukee
10. Minneapolis	11. Pierre	12. Bismarck
13. Helena	14. Seattle	15. Portland
16. Boise	17. Salt Lake City	18. Carson City
19. Los Angeles	20. Phoenix	21. Santa Fe
22. Denver	23. Cheyenne	24. Omaha
25. Des Moines	26. Kansas City	27. Topeka
28. Oklahoma City	29. Dallas	30. Little Rock
31. Memphis	32. Jackson	33. New Orleans
34. Birmingham	35. Atlanta	36. Jacksonville
37. Columbia	38. Raleigh	39. Richmond
40. Washington D.C.	41. Boston	42. Portland

[1] [3]

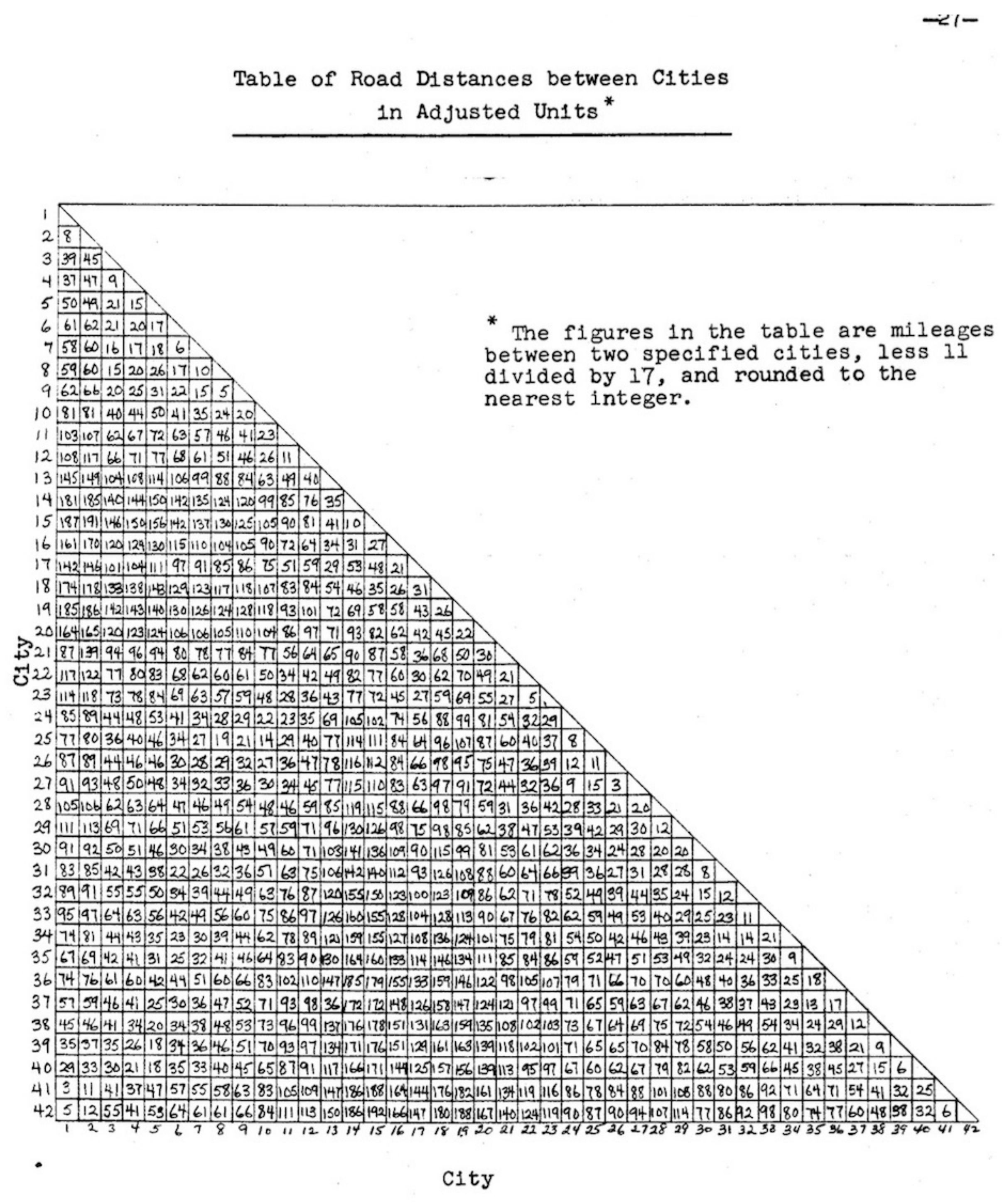


Table of Road Distances between Cities in Adjusted Units*

* The figures in the table are mileages between two specified cities, less 11 divided by 17, and rounded to the nearest integer.

City

City	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
1																																										
2	8																																									
3	39	45																																								
4	37	47	9																																							
5	50	49	21	15																																						
6	61	62	21	20	17																																					
7	58	60	16	17	18	6																																				
8	59	60	15	20	26	17	10																																			
9	62	66	20	25	31	22	15	5																																		
10	81	81	40	44	50	41	35	24	20																																	
11	103	107	62	67	72	63	57	46	41	23																																
12	108	117	66	71	77	68	61	51	46	26	11																															
13	145	149	104	108	114	106	99	88	84	63	49	40																														
14	181	185	140	144	150	142	135	124	120	99	85	76	35																													
15	187	191	146	150	156	142	137	130	125	105	90	81	41	10																												
16	161	170	120	129	130	115	110	104	105	90	72	64	34	31	27																											
17	142	146	101	104	111	97	91	85	86	75	51	59	29	53	48	21																										
18	174	178	133	138	143	129	123	117	118	107	83	84	54	46	35	26	31																									
19	185	186	142	143	140	130	126	124	128	118	93	101	72	69	58	58	43	26																								
20	164	165	120	123	124	106	106	105	110	104	86	97	71	93	82	62	42	45	22																							
21	87	139	94	96	94	80	78	77	84	77	56	64	65	90	87	58	36	68	50	30																						
22	117	122	77	80	83	68	62	60	61	50	34	42	49	82	77	60	30	62	70	49	21																					
23	114	118	73	78	84	69	63	57	59	48	28	36	43	77	72	45	27	59	69	55	27	5																				
24	85	89	44	48	53	41	34	28	29	22	23	35	69	105	102	74	56	88	99	81	54	32	29																			
25	77	80	36	40	46	34	27	19	21	14	29	40	77	114	111	84	64	96	107	87	60	40	37	8																		
26	87	89	44	46	46	30	28	29	32	27	36	47	78	116	112	84	66	98	95	75	47	36	39	12	11																	
27	91	93	48	50	48	34	32	33	36	30	34	45	77	115	110	83	63	97	91	72	44	32	36	9	15	3																
28	105	106	62	63	64	47	46	49	54	48	46	59	85	119	115	88	66	98	79	59	31	36	42	28	33	21	20															
29	111	113	69	71	66	51	53	56	61	57	59	71	96	130	126	98	75	98	85	62	38	47	53	39	42	29	30	12														
30	91	92	50	51	46	30	34	38	43	49	60	71	103	141	136	109	90	115	99	81	53	61	62	36	34	24	28	20	20													
31	83	85	42	43	38	22	26	32	36	51	63	75	106	142	140	112	93	126	108	88	60	64	66	39	36	27	31	28	28	8												
32	89	91	55	55	50	34	39	44	49	63	76	87	120	155	150	123	100	123	109	86	62	71	78	52	49	39	44	35	24	15	12											
33	95	97	64	63	56	42	49	56	60	75	86	97	126	160	155	128	104	128	113	90	67	76	82	62	59	49	53	40	29	25	23	11										
34	74	81	44	43	35	23	30	39	44	62	78	89	121	159	155	127	108	136	124	101	75	79	81	54	50	42	46	43	39	23	14	14	21									
35	67	69	42	41	31	25	32	41	46	64	83	90	130	164	160	133	114	146	134	111	85	84	86	59	52	47	51	53	49	32	24	24	30	9								
36	74	76	61	60	42	44	51	60	66	83	102	110	147	185	179	155	133	159	146	122	98	105	107	79	71	66	70	70	60	48	40	36	33	25	18							
37	57	59	46	41	25	30	36	47	52	71	93	98	36	172	172	148	126	158	147	124	121	97	99	71	65	59	63	67	62	46	38	37	43	23	13	17						
38	45	46	41	34	20	34	38	48	53	73	96	99	137	176	178	151	131	163	159	135	108	102	103	73	67	64	69	75	72	54	46	49	54	34	24	29	12					
39	35	37	35	26	18	34	36	46	51	70	93	97	134	171	176	151	129	161	163	139	118	102	101	71	65	65	70	84	78	58	50	56	62	41	32	38	21	9				
40	29	33	30	21	18	35	33	40	45	65	87	91	117	166	171	144	125	157	156	139	113	95	97	67	60	62	67	79	82	62	53	59	66	45	38	45	27	15	6			
41	3	11	41	37	47	57	55	58	63	83	105	109	147	186	188	164	144	176	182	161	134	119	116	86	78	84	88	101	108	88	80	86	92	71	64	71	54	41	32	25		
42	5	12	55	41	53	64	61	61	66	84	111	113	150	186	192	166	147	180	188	167	140	124	119	90	87	90	94	107	114	77	86	92	98	80	74	77	60	48	38	32	6	

City

Figure 6: Distance Matrix for Dantzig 42

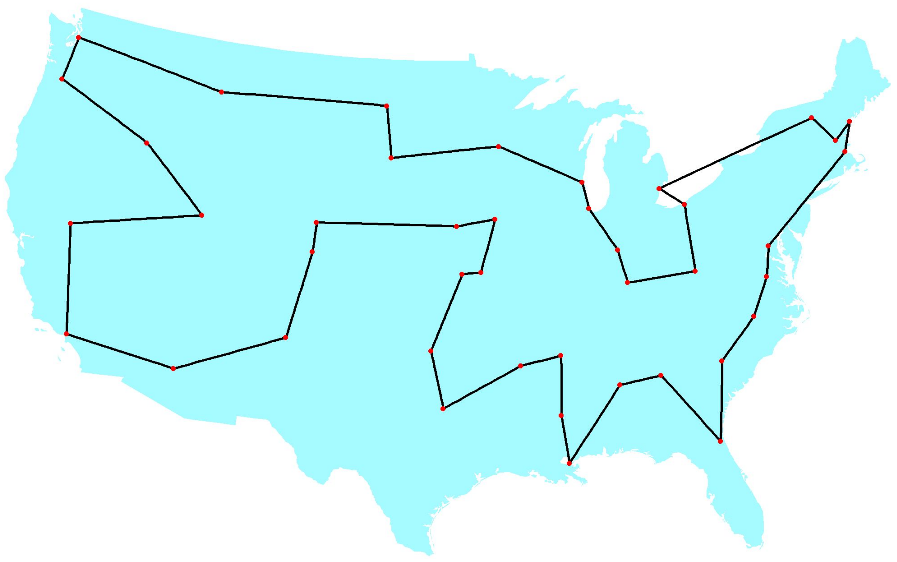
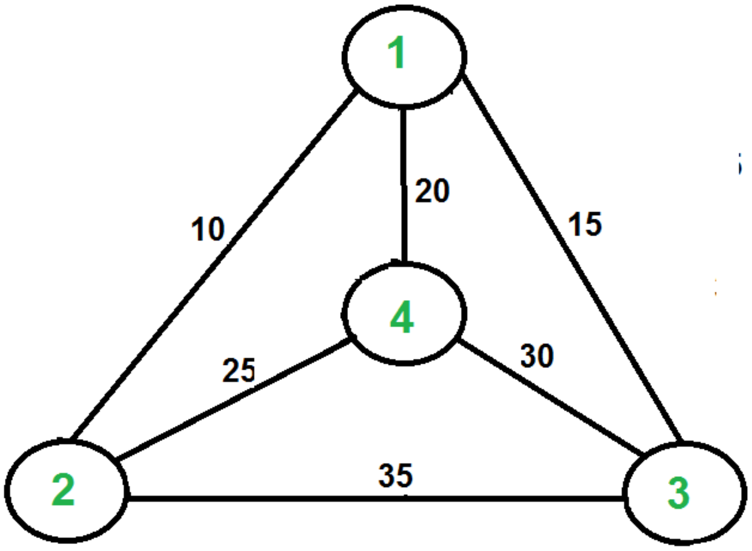


Figure 7: Dantzig 42 optimal solution

3.3 Dynamic Programming

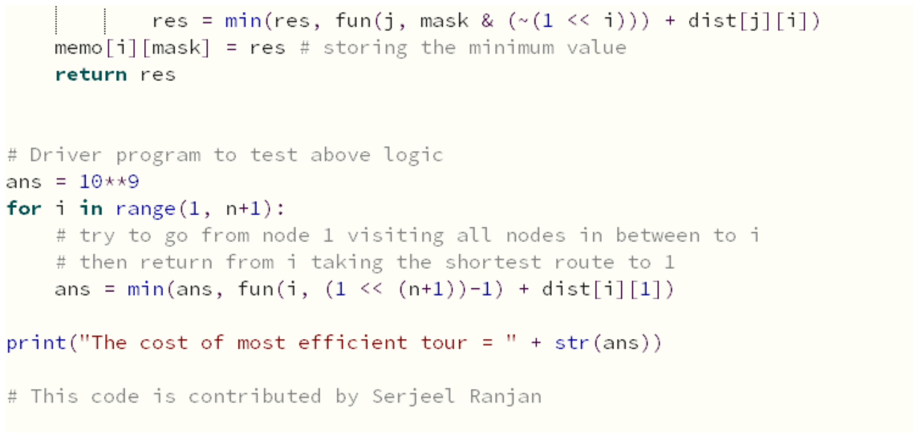
We have a set of vertices numbered from 1 to n. We want to find the shortest cycle that starts and ends at vertex 1, visiting each vertex exactly once. To do this, we calculate the minimum cost path from vertex 1 to each other vertex, and then add the distance from that vertex back to vertex 1. We repeat this process for each vertex and return the minimum total cost.

To find the cost for each vertex, we use dynamic programming. We define a function C(S, i) as the cost of the minimum path starting at vertex 1, ending at vertex i, and visiting each vertex in set S exactly once. We start by calculating C(S, i) for all subsets S of size 2, then for all subsets of size 3, and so on, until we cover all vertices. In each step, we ensure that vertex 1 is included in the subset.



The python code for this example looks as following:

```python
n = 4 # there are four nodes in example graph (graph is 1-based)

# dist[i][j] represents shortest distance to go from i to j
# this matrix can be calculated for any given graph using
# all-pair shortest path algorithms
dist = [[0, 0, 0, 0, 0], [0, 0, 10, 15, 20], [
    0, 10, 0, 25, 25], [0, 15, 25, 0, 30], [0, 20, 25, 30, 0]]

# memoization for top down recursion
memo = [[-1]*(1 << (n+1)) for _ in range(n+1)]


def fun(i, mask):
    # base case
    # if only ith bit and 1st bit is set in our mask,
    # it implies we have visited all other nodes already
    if mask == ((1 << i) | 3):
        return dist[1][i]

    # memoization
    if memo[i][mask] != -1:
        return memo[i][mask]

    res = 10**9 # result of this sub-problem

    # we have to travel all nodes j in mask and end the path at ith node
    # so for every node j in mask, recursively calculate cost of
    # travelling all nodes in mask
    # except i and then travel back from node j to node i taking
    # the shortest path take the minimum of all possible j nodes
    for j in range(1, n+1):
        if (mask & (1 << j)) != 0 and j != i and j != 1:
```



```
            res = min(res, fun(j, mask & (~(1 << i))) + dist[j][i])
    memo[i][mask] = res # storing the minimum value
    return res


# Driver program to test above logic
ans = 10**9
for i in range(1, n+1):
    # try to go from node 1 visiting all nodes in between to i
    # then return from i taking the shortest route to 1
    ans = min(ans, fun(i, (1 << (n+1))-1) + dist[i][1])

print("The cost of most efficient tour = " + str(ans))

# This code is contributed by Serjeel Ranjan
```

[8]

4 Modern Day Scenarios

4.1 Transportation

The traveling salesman problem (TSP) is extremely useful in the modern-day, with its most obvious use case being for companies in the transportation industry. These companies often need to deliver products to several places in the most efficient way, both to save on energy consumption and for customer convenience. [4]

4.2 Telecommunications

Another important use case for the TSP is in the telecommunications industry. To best satisfy customers, telecommunications companies need to implement their stations in such a way as to provide network access to the most amount of people while still retaining a high-performance connection. [4]

4.3 Manufacturing

The TSP is additionally useful for manufacturers. In factories, machines are arranged in a way that allows for a worker to quickly reach one in case it needs maintenance; having the shortest possible distance between all parts of the production line allows for the factory to be most productive, as less time is wasted on fixing it. [4]

4.4 Bio-engineering

The traveling salesman problem is also important in biology, specifically in dealing with DNA sequencing. When gathering DNA fragments, scientists can use the TSP to figure out how to reconstruct them back to their original sequence. [4]

4.5 Robotics

Recently, there has been an influx of use cases for the TSP in machinery. For instance, self-driving cars require knowledge of the most optimal paths on the road to maximize efficiency. Moreover, robots such as Roombas need to be able to move most efficiently in a given area to conserve energy. [4]

5 Solution for in class nearest neighbor problem

Nearest Neighbor Solution for node 3: 3 - 4 - 6 - 5 - 2 - 1 - 3

$$2.08 + 0.97 + 3.01 + 4.06 + 2.21 + 2.34 = 14.67$$

Optimal solution: 1 - 2 - 4 - 6 - 5 - 3 - 1

$$2.21 + 2.12 + 0.97 + 3.01 + 2.39 + 2.34 = 13.04$$

6 Sources Cited

References

[1] S Johnson G Dantzig R Fulkerson. "Solution of a Large-Scale Travelling-Salesman Problem". In: *The Rand Corporation* (1954). URL: https://www.hexaly.com/wp-content/uploads/2022/04/TSP_MIP.pdf?redirect.

[2] *On the history of combinatorial optimization (till 1960)*. 1960. URL: https://homepages.cwi.nl/~lex/files/histco.pdf (visited on 04/26/2024).

[3] *Table of Road Distances Between Cities in Adjusted Units*. University of Waterloo. URL: https://www.math.uwaterloo.ca/tsp/us/img/dfj_table.jpg (visited on 04/26/2024).

[4] *The Traveling Salesman Problem in Manufacturing*. Optessa. 2024. URL: https://www.optessa.com/the-traveling-salesman-problem/#:~:text=Despite%20the%20challenges%2C%20the%20TSP,can%20significantly%20impact%20these%20fields. (visited on 02/23/2024).

[5] *Traveling Salesman Problem: Nearest Neighbor Algorithm Solution*. medium. 2023. URL: https://blog.devgenius.io/traveling-salesman-problem-nearest-neighbor-algorithm-solution-e78399d0ab0c.

[6] *Travelling Salesman Problem Intro - Brute Force Method*. Antony Joms. 2020. URL: https://www.youtube.com/watch?v=R_wLqwTuDzU (visited on 2020).

[7] *Travelling Salesman Problem Intro - Brute Force Method*. Ethan Li. 2024. URL: https://github.com/ethan201not404/som-discovery-topic-research-tsp/tree/main/PokemonMap (visited on 2024).

[8] *Travelling Salesman Problem using Dynamic Programming*. geeksforgeeks. 2023. URL: https://www.geeksforgeeks.org/travelling-salesman-problem-using-dynamic-programming/ (visited on 04/19/2023).

https://www.math.uwaterloo.ca/tsp/history/index.html