THE TRAVELLING SALESMAN PROBLEM

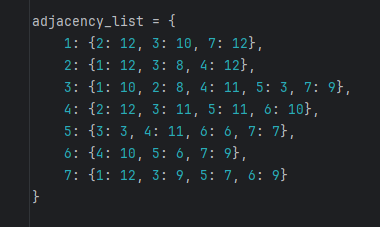
This is a real-world problem that involves a fixed set of points, all must be traversed and return to the starting point, without reaching a point twice and taking the shortest route possible.

Input: Data structure

Points are stored as a dictionary where the;

Keys: All the individual points.

Values: Dictionaries containing the keys as cities adjacent to the key(point) and their respective distances from the current node.



Effectiveness

It is easy to check validity of a path by checking if the next node is in the adjacent points of that point.

Every adjacent point has a distance from the current node tied to it so a direct reference can be made.

Output: A sequence of numbers representing the best optimal route.

Solution;

Two types of solutions (algorithms) are available;

Exact (Non-Heuristic)

These guarantee a valid best path.

Heuristic (Approximate)

These do not guarantee the best path or a valid one.

Heuristic

Nearest-Neighbour

Genetic Algorithm

Simulated Anealing

Christofides Algorithm (Best approximation technique)

Self-Organising-Maps

Exact

Brute-force (Exhaustive Search)

Dynamic Programming (Held-Karp)

Branch and Bound

Comparison of solutions

Algorithms are classified mainly by complexity (number of steps), performance and how much of a guarantee they offer for an optimal solution.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Algorithm** | **Description** | **Complexity** | Suitability | **Pros** | **Cons** |
| **Dynamic Programming** | Solves optimally by breaking the problem into subproblems | *O*(*n*2⋅2*n*) | 20-30 cities | Optimal solution, avoids recomputation | Memory-heavy, computationally expensive for large n |
| **Branch-and-Bound** | Systematically explores the solution space with pruning | O(e^n) | problems requiring optimal solution | Guarantees optimal solution if fully explored  Prunes parts of the search space to avoid unnecessary checks. | Still computationally expensive, impractical for large datasets |
| **Nearest Neighbor** | Greedy heuristic; picks nearest city next | *O*(*n*2) | Large problems needing quick solutions | Fast, simple to implement | No optimality guarantee, can get stuck in local minima |
| **Self-Organizing Map (SOM)** | Neural network clustering approach, useful for approximation | Varies (depends on grid size and iterations) | Large problems, | Scalable, good for large datasets higer dimension, captures patterns | No guarantee of optimality, solution quality depends on map configuration |
| Brute-force | Tests all possible solutions and picks the best. | *O*(*n*!) | < 10 points | Always guarantees optimal path,  Easy to implement | factorial time complexity increases extremely rapidly with n |
| **Christofides Algorithm** | the direct distance between two cities is always less than or equal to the sum of the distances through a third city | *O*(*n*3) | where the triangle inequality holds | Guarantees a 1.5 times approximation to the optimal.  Easy to implement.  Most accurate approximation. | No guarantee of optimality, solution |
| **Genetic Algorithm** | Depends on population size and generations | Depends on population size and generations | When an exact solution is not needed. | Provides a near optimal  approximation | No guarantee of optimality, solution  Implementation is hectic |
| **Simulated Annealing** | Probabilistic and depends on temperature decay and iterations | Depends on temperature decay and iterations | When an exact solution is not needed. | Less computational since they path with highest probability is considered. | No guarantee of optimality, solution  Implementation is hectic |

Example showing complexity versus number of points.

| **n*n* (Number of Cities)** | **Brute Force O(n!)*O*(*n*!)** | **Dynamic Programming O(n2⋅2n)*O*(*n*2⋅2*n*)** | **Nearest Neighbor O(n2)*O*(*n*2)** | **Christofides O(n3)*O*(*n*3)** | **Branch and Bound (Exponential)** |
| --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 2 | 4 | 4 | 8 | 2 |
| 3 | 6 | 36 | 9 | 27 | 6 |
| 4 | 24 | 384 | 16 | 64 | 24 |
| 5 | 120 | 4,800 | 25 | 125 | 120 |
| 6 | 720 | 61,440 | 36 | 216 | 720 |
| 7 | 5,040 | 823,680 | 49 | 343 | 5,040 |
| 8 | 40,320 | 10,485,760 | 64 | 512 | 40,320 |
| 9 | 362,880 | 134,217,728 | 81 | 729 | 362,880 |
| 10 | 3,628,800 | 1,717,986,918,400 | 100 | 1,000 | 3,628,800 |

According to the statistical facts Exhaustive Search (Brute-Force) will be used because;

It guarantees an exact, optimal path.

Easy to implement.

Practical for this dataset n = 7 points.

Can be optimized to use up less memory easily.

How it works

Rule 1: Visit all points.

Find all numbers with n-1 digits without 1 in them.

Rule 2: Visit each point exactly once.

Eliminate all numbers with any repeating digits e.g 223457

Rule 3: Start and end at the same point.

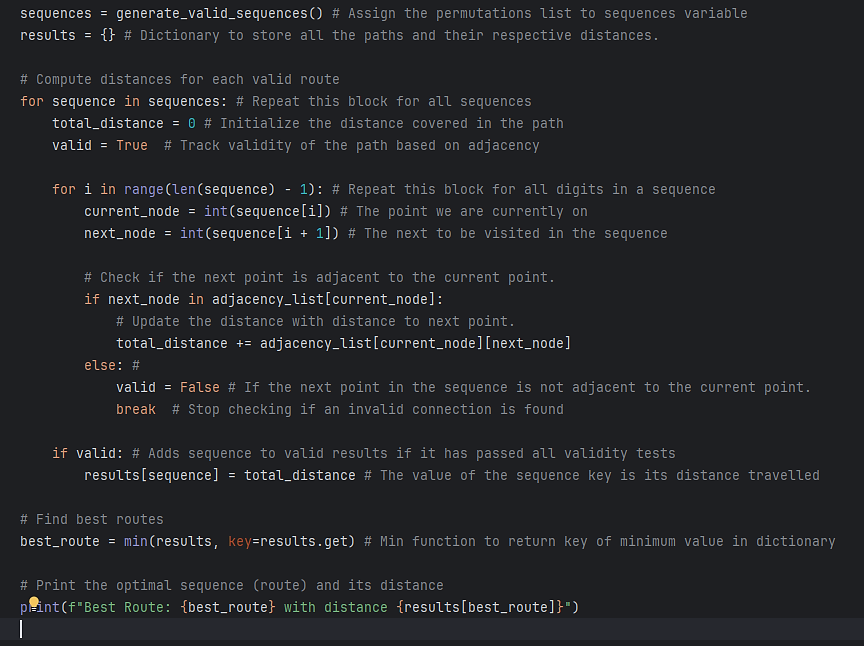
Add 1 at the beginning and end of each number obtained to obtained all the possible paths.

Rule 4: Shortest distance possible.

Use the adjacency list to calculate the distance of the current digit (current node) from the next digit in the sequence (next node) while checking that the next node is adjacent to the current node. If not, that path is discarded.

Source code: 





Self-Organising-Maps (SOM’s) / Kohonen Maps

Unsupervised neural network that clusters high dimensional data through creating a discretized version of its input space(map) in lower dimensions.

Created by Teuvo Kohonen

They use competitive learning instead of error-correcting learning by using a function related to neighbourhoods to preserve the topological properties of the map.

They consist of the input layer, the weights and the output(Kohonen/feature/competitive layer)

They are great for dimensional reduction, clustering, natural language processing, geology, astronomy and finance.

Weights – Coordinates of each output neuron.

Output neurons must compete to be activated, hence become the winner.

The learning scheme is comprised of;

Initialization

Data structures, samples and training data is loaded

Competition

Winner is determined

Cooperation

Topological neighbor of winner is determined

Adaptation

The weights are updated to increase importance of neurons is updated

Continuation

Algorithm is repeated for a given number of iterations

Phases of the learning process

Initialization of weights with random values

Sampling

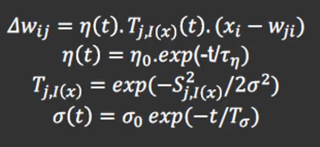
Go through dataset and select samples randomly in each iteration from map

Matching

Finding winner (with weight vector closest (shortest eucledian distance) to the input sample) and its topological neighbours

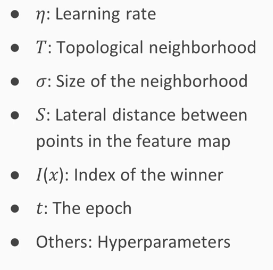
Updating

Changing neuron weights of the winner and its topological neighbours based on a specific equation



Where;

Dw – weight change of the neuron.



Topological neighbourhood and learning rate decay as iterations increase.

Continuation / Convergence

Repeat the same algorithm until the feature map remains the same or until the number of iterations(epoch t) is reached.

Illustration of feature(output) map showing winning neuron and its topological neighbourhood

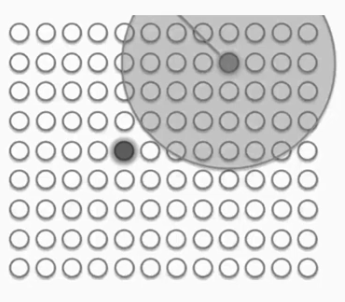
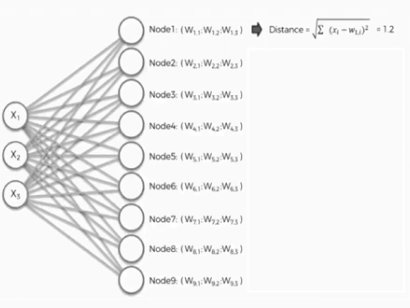


Illustration of how mapping occurs



The weights are similar to the inputs of the input space.

Step by step, the winning neuron is mapped onto the feature map until the whole map is completely covered or all neurons are mapped.

Training results

Limitations and difficulties

Data must be converted to standard .tsp format for input into the algorithm.

Implementation is not straight forward.

Slow and computationally expensive (not suitable for small datasets).

Comparison

Rate quality

Exhaustive Search is shorter to implement, run and adopt as opposed to SOM that needs multiple files, equations, functions and processes. Speed can be reduced by reducing number of iterations

[add screenshots of speed measurements in console]

complexity

Exhaustive search is more complex with steps equal to 7! = 5040 while SOM complexity depends on the number of iterations used.

Time complexity

high-level discussion of the computational cost of the SOM approach (number of iterations, updates per iteration, compare with exhaustive.

Consider a scenario of a mailman charged with delivering mail in Kampala.

Continue

Extensions