Logo

Description automatically generated with low confidence

**Faculty of Science and Technology**

Department of Computer Science

Chart

Description automatically generated

Travelling Salesman Problem

CST3170 ARTIFICIAL INTELLIGENCE

STUDENT NAME: OMER KACAR

STUDENT NUMBER: M00719709

MODULE LEADER: CHRIS HUYCK

CAMPUS: HENDON

Contents

[1 Introduction 3](#_Toc90306915)

[2 Genetic Algorithm 3](#_Toc90306916)

[2.1 Steps 3](#_Toc90306917)

[2.2 Threads 3](#_Toc90306918)

[2.3 Results of All Four Tests 3](#_Toc90306919)

[2.4 Conclusion 4](#_Toc90306920)

[3 Dijkstra 4](#_Toc90306921)

[3.1 Steps 4](#_Toc90306922)

[3.2 Priority Queue: 4](#_Toc90306923)

[3.3 Conclusion 4](#_Toc90306924)

[4 Breadth First Search 4](#_Toc90306925)

[4.1 Steps 4](#_Toc90306926)

[4.2 Queue 5](#_Toc90306927)

[4.3 Conclusion 5](#_Toc90306928)

[5 Heuristic Nearest Neighbours 5](#_Toc90306929)

[5.1 Steps 5](#_Toc90306930)

[5.2 Heuristic 5](#_Toc90306931)

[5.3 Conclusion 5](#_Toc90306932)

[6 Permutation 5](#_Toc90306933)

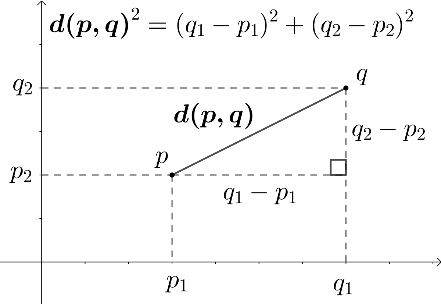
[6.1 Steps 5](#_Toc90306934)

[6.2 Conclusion 5](#_Toc90306935)

[7 Final Note 6](#_Toc90306936)

# Introduction

The paper provides an approach based on learning for approximately solving the Travelling Salesman Problem on 2D Euclidean graphs (Figure 1).

Figure 1

*“Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city?"*

There are several integer constrained optimisation problems that are intractable to solve optimally at very large scales known as NP-hard problems. And thus, appropriate approximation algorithms to popular problems with NP-hardness have a variety of practical applications and are crucial in modern industries such as transportation, logistics, finance, and energy.

This paper examines approximation algorithms such as the genetic algorithm and other approaches, such as Dijkstra's, breadth first search, nearest neighbour heuristics and the permutation algorithm.

# Genetic Algorithm

The best algorithm attempted is the genetic algorithm.

## Steps

1. Begins with creating a generation of 50 genes, for which, each gene in the generation contains randomised indexes of possible paths and the length of a gene is the total number of cities.
2. Chooses the best gene from the generation of genes. This is called the parent and has the shortest path in the generation.
3. A parent produces 50 offspring. Each offspring has parts from the same parent. The offspring differ based on the crossover point exchanged from the parent.
4. Every index of an offspring will have a 15% chance of being mutated.
5. As a result of the mutation, the path is modified by switching indexes between two cities (one is the mutated index and the other is the randomised index).
6. The steps above create one generation of 50 genes, this is then repeated 1000 times.

## Threads

There is a maximum of 5000 threads, in which, the genetic algorithm is rerun completely for each thread.

## Results of All Four Tests

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Number** | **Optimal Path** | **Distance** | **Time Elapsed** |
| 1 | 4 2 12 8 9 1 7 11 6 10 3 5 4 | 275.583 | 44 ms |
| 2 | 8 11 9 14 1 4 2 13 3 12 10 6 7 5 8 | 836.792 | 57 ms |
| 3 | 12 14 9 3 17 1 8 13 11 2 6 7 10 5 4 15 16 12 | 121052.340 | 62 ms |
| 4 | 19 13 5 8 11 2 4 12 17 20 22 1 15 10 14 25 24 18 23 7 6 26 21 16 9 27 3 19 | 2087562.212 | 77 ms |

Below is a table of the results for the different data files. It lists the optimal path found for the given city coordinates, the distance along the path, and finally the time to find the result

## Conclusion

The time complexity of the genetic algorithm is , for which, is the number of threads to run, and is the creation of genes and the evaluation of each index of a gene. The number of executions for a thread is greater, so the time complexity . This multi-threaded genetic algorithm consistently produces optimal results for the first three test files. But for the fourth test file, the results varied (but not by much) since there were 27 cities. Be that as it may, the genetic algorithm is ran multiple times through multi-threading, which makes it sufficiently detailed and creative.

# Dijkstra

## Steps

1. Begins with a starting index, this is considered a path.
2. Adding new points to the initial path creates new paths for each new point.
3. Then, these paths are added to the priority queue.
4. The path with the shortest distance has a higher priority in the queue. Now, the path with the shortest distance will always be in front of the queue until the path to all the cities is found.

## Priority Queue:

* A doubly linked list, performing all the necessary functions of a queue (enqueue, dequeue).
* The enqueue time complexity is , for which, *n* is the iteration through each element (path) in the queue and *m* is the calculation of the total distance of each path. And thus, the time complexity is . The time complexity is ‘ because the queue is going to have more elements than the number of elements in a path (for calculating the distance).
* The dequeue method takes time since only the element at the front of the list is removed.

## Conclusion

In Dijkstra's case, the priority queue performs many iterations for processes such as enqueue (i.e., traversing the linked list, calculating distances). Hence, this Dijkstra's algorithm is very slow and did not outperform the breadth first search approach.

# Breadth First Search

## Steps

1. Begins with a starting index, this is considered a path.
2. This is followed by exploring all indices at the current depth level before progressing to the next possible indices to add to the path.
3. Upon reaching the depth level containing a path to every city, the distances of the paths will be compared, and the shortest distance path will be chosen.

## Queue

* Uses a two-dimensional array to store the paths. This means each element in the queue is an array. In this case, the space used is more efficient than Dijkstra’s doubly linked list that refers to two objects and contains an array (which is the path).
* The time complexity of enqueuing and dequeuing is because to remove the front element of the queue or to add a new element to the end takes one time.
* When an array is at its maximum size, it can take time to enqueue or dequeue an element since the array must copy all its elements into a new two-dimensional array with double the capacity.

## Conclusion

Breadth first search is sufficient for finding the optimal path for up to 10 cities. However, there are too many possible directions and paths to find the optimal path when there are more than 10 cities, and thus, breadth first search can cause out of memory errors because of Java’s limited heap space size.

# Heuristic Nearest Neighbours

## Steps

1. Begins with the starting city in the data file.
2. Each time a city is visited, the next city to visit is its closest neighbouring city.
3. Repeat step 2 from the next city until all the cities have been visited.
4. In order to create a cycle, the last city visited is linked with the first city.
5. Repeat steps 1 to 4 but start from the next city instead.

## Heuristic

When each nearest neighbour algorithm is completed, the complete path's distance is compared with the distance of previous paths (which started from a different city). As a result, the shortest path can be updated.

The time complexity of the nearest neighbour algorithm is , for which, is the number of possible starting points and is the search for the nearest neighbour.

## Conclusion

Although the heuristic nearest neighbour algorithm is fast, it does not find the shortest route in all cases. This is largely since getting the closest point from each point is not a reliable method of determining the shortest path.

# Permutation

## Steps

1. Starts from an array of consecutive integers representing the cycle to all the cities.
2. Utilise recursion to perform a permutation of this path until all possible paths are found.

## Conclusion

Since the number of possible paths is -factorial ( being the number of cities), after 11 cities, the permutation method is not a viable solution. As the array size increases, it will exceed the maximum size of an array or Java will eventually run out of heap space during the search for all the possible paths. The time complexity for the permutation algorithm is .

# Final Note

Algorithms such as heuristic nearest neighbour performed well but did not guarantee optimal path. While the permutation method, breadth first search, and Dijkstra's algorithm produced optimal results, they were too exhaustive and were not suitable for a larger number of cities. With the given number of cities in the test files, the genetic algorithm was the best and fastest option. A limitation was the code to read in the data took 30 milliseconds to run, which added 30 milliseconds to the competition's time.