# Project 1: Traveling Salesman Problem – Brute Force

Nick York
CES Department
Speed School of Engineering
University of Louisville, USA
neyork01@louisville.edu

## 1. Introduction

The Traveling Salesman Problem ("the TSP") requires finding the minimum cost route ("optimal route") for a salesman to visit a list of cities without visiting a city more than once, except for the starting city. The salesman must return to their starting city at the end. Each city is represented by an xy-coordinate pair and numbered starting at one. The coordinates are provided by a .tsp file ("TSP file"). The cost to travel between two cities is the Euclidean distance between them, calculated as:

$$cost = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

More than one optimal route is possible; for instance, the reverse of an optimal route has the same cost and therefore is optimal. The goal of this project was to write a program ("the program") to find the optimal route for a given set of cities using brute force.

## 2. Approach

I wrote a Python program that allows the user to specify which TSP file(s) to use. Next, the program reads and parses the xy-coordinate pairs in a TSP file and passes them as an argument to the *solve_TSP* function. *solve_TSP* calls a helper function to build a matrix listing the cost between each possible pair of cities to avoid repeating calculations. The last step before finding the optimal route is to generate all permutations of cities ending at the starting city. I used the *permutations* function from the standard library module called itertools to generate the permutations.

The *permutations* function takes a list of elements and returns all permutations of them as a generator, a Python object that can be iterated over to retrieve each value.[1] Internally, the function uses two list variables: indices, composed of numbers from 0 to n-1 (where n is the length of the argument) and cycles, composed of numbers from n to 1. Then, a for loop iterates over all possible permutation cycles, swapping elements of cycles to build each permutation using an if-else

[1] Initially I converted the generator to a list before using it. However, storing large permutations as a list caused the program to run out of memory. Generators are a memory efficient means of processing large amounts of data.

statement. I iterated over the generator to retrieve each permutation and appended the permutation's first element to itself to create each route.

Two variables, min_cost and optimal_route, track the current minimum cost and optimal route. Both variables are initialized to infinity and None, respectively. Each iteration finds the cost of the current route using the cost matrix and compares it with min_cost. If the cost is less than min_cost, then the cost and current route are assigned to min_cost and optimal_route. Once all routes have been iterated over, the final value of min_cost and optimal_route is the solution to the TSP.

# 3. Results

## 3.1 Data

For this project the students were provided several TSP files with xy-coordinate pairs representing cities to test our program on. Each TSP file has an increasing number of cities.

## 3.2 Results

The results for each TSP file is printed to the console during runtime (Fig 1). The optimal route for each TSP file is also plotted and saved as a PNG file under a folder named Results (Fig 2).[2] Likewise, the results for each TSP file are also saved to a CSV file under Results, listing the TSP filename, the optimal route, cost, and runtime in seconds (Fig 3). The runtime is less than a second for 4 cities until 9 cities and above, after which it increases dramatically (Fig 4).

```
Note: All routes end at the starting city.

The optimal route for Random4: [1, 3, 2, 4, 1]
Cost: 215.085533

The optimal route for Random5: [1, 2, 5, 3, 4, 1]
Cost: 139.133542

The optimal route for Random6: [1, 2, 3, 4, 5, 6, 1]
Cost: 118.968914

The optimal route for Random7: [1, 2, 7, 3, 6, 5, 4, 1]
Cost: 63.863031

The optimal route for Random8: [1, 6, 8, 4, 5, 2, 3, 7, 1]
Cost: 310.98208

The optimal route for Random9: [1, 7, 6, 3, 5, 2, 9, 4, 8, 1]
Cost: 131.028366

The optimal route for Random10: [1, 2, 7, 6, 8, 5, 9, 10, 4, 3, 1]
Cost: 106.78582

The optimal route for Random11: [1, 2, 4, 3, 5, 7, 9, 8, 11, 10, 6, 1]
Cost: 252.684435

The optimal route for Random12: [1, 8, 2, 3, 12, 4, 9, 5, 10, 6, 7, 11, 1]
Cost: 66.084844
```

*Fig 1. Output of running the program to solve all TSP files in the Files directory.*

[2] I adapted my plotting solution from an answer to a StackOverflow question asking how to draw arrows on a plot (see References for link).

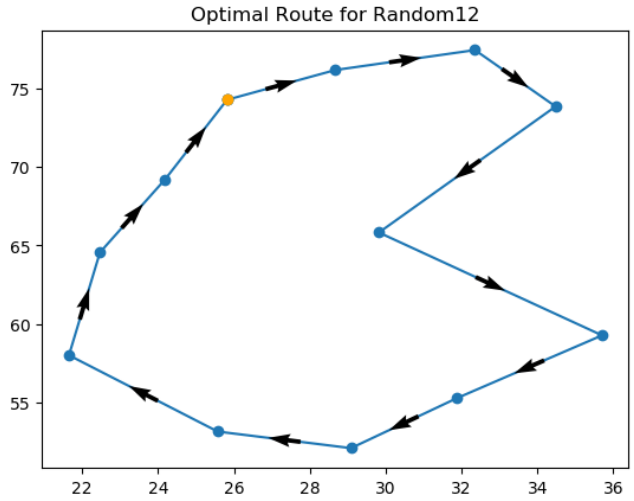


*Fig 2. Optimal route for Random12.tsp with arrows drawn between coordinates and the starting coordinate colored orange.*

| File | Optimal Route | Cost | Running Time (sec) |
|---|---|---|---|
| Random4.tsp | [1, 3, 2, 4, 1] | 215.085533 | 0.761069536 |
| Random5.tsp | [1, 2, 5, 3, 4, 1] | 139.133542 | 0.236765862 |
| Random6.tsp | [1, 2, 3, 4, 5, 6, 1] | 118.968914 | 0.194526672 |
| Random7.tsp | [1, 2, 7, 3, 6, 5, 4, 1] | 63.863031 | 0.226425886 |
| Random8.tsp | [1, 6, 8, 4, 5, 2, 3, 7, 1] | 310.98208 | 0.633155584 |
| Random9.tsp | [1, 7, 6, 3, 5, 2, 9, 4, 8, 1] | 131.028366 | 4.26855135 |
| Random10.tsp | [1, 2, 7, 6, 8, 5, 9, 10, 4, 3, 1] | 106.78582 | 49.36330533 |
| Random11.tsp | [1, 2, 4, 3, 5, 7, 9, 8, 11, 10, 6, 1] | 252.684435 | 538.8870969 |
| Random12.tsp | [1, 8, 2, 3, 12, 4, 9, 5, 10, 6, 7, 11, 1] | 66.084844 | 6497.991282 |

*Fig 3. Results for each TSP file provided by the professor.*

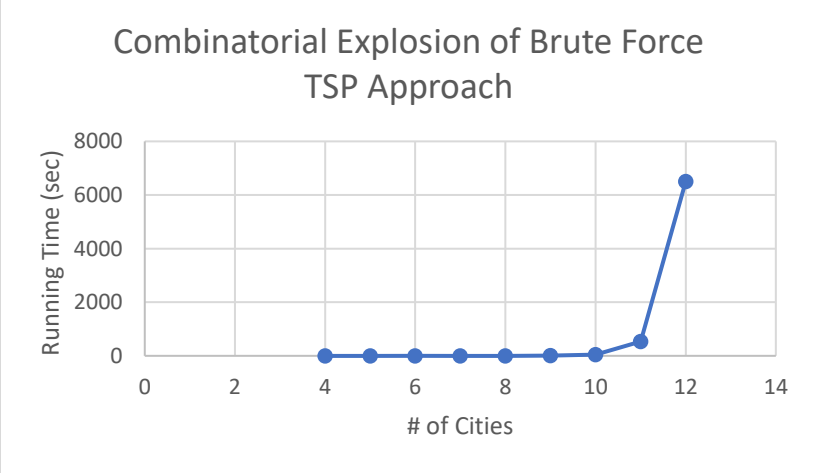


*Fig 4. Plot of # of cities versus running time (in seconds) demonstrating combinatorial explosion.*

## Discussion

A brute force approach to the TSP works for a small number of cities. However, this approach is susceptible to combinatorial explosion. Fig 4 demonstrates how an increase of one city (from 11 to 12) causes the running time to increase exponentially. The reason for this is that *permutations* performs n! iterations to generate all permutations, where n is the number of elements. The program iterates n! times over the generator returned by *permutations* and iterates an additional n times over each permutation. The program's time complexity works out to $O(n \cdot n!)$. For a modest number of cities, the brute force approach to the TSP becomes intractable.

## References


The Python Standard Library – Functional Programming Modules – Itertools – Permutations

StackOverflow answer to "Show direction arrows in a scatterplot"