## Brute-force Solution

### Description

The brute-force solution is an exact solution, but with an extremely poor runtime. Put simply, given a starting vertex *s*, a brute-force algorithm will look at every potential path beginning and ending at *s*, ultimately returning the smallest path that it encountered. Another way to think of the potential paths is as permutations of the vertices to be visited, with the restriction that the permutations begin and end at *s*. [1]

However, the number of permutations of vertices is extremely large – O(*n*!), where *n* represents the number of cities on the map – so the brute-force algorithm becomes impractical even at an *n* of only 20. [2] Regardless of its impracticality at large values of *n*, the brute-force method is an exact algorithm, so it is worth considering as a baseline for the efficiency and accuracy of our other exact solution, branch and bound (discussed in the subsequent section).

### Pseudocode

The basic pseudocode for the brute-force solution is as follows:

find an initial Hamiltonian tour, called T

set the min tour to T

set the min distance to T.distance

while there are unchecked permutations of T, excluding start and end vertex *s*:

generate a new permutation of T, called T’

if T’.distance < min distance:

set the min tour to T’

set the min distance to T’.distance

[1]

The process of getting permutations is where the complexity is introduced, as there are O(*n*!) permutations of *n* cities. When getting permutations, we would also need to be careful to exclude the start/end vertex *s* from the permutation, since those points will never change. Thus, more specifically, there are (*n* – 1)! permutations of a path from *s* back to *s*, since *s* is excluded from the permutation process. We still need to take into account the distance from *s* to other vertices, though, so we can’t simply remove it from the calculations.

Simple pseudocode for generating the permutations is as follows:

permutations = []

permute(T, start, end):

if start == end:

permutations.add(T)

else:

for i from start to end:

swap(T[start], T[i])

permute(T, start+1, end);

swap(T[start], T[i]) // backtrack

[3]

# Bibliography

|  |  |
| --- | --- |
| [1] | T. Donaldson, "13. Case Study: Solving the Traveling Salesman Problem," 2012. [Online]. Available: http://www.cs.sfu.ca/CourseCentral/125/tjd/tsp\_example.html. [Accessed 31 May 2018]. |
| [2] | "Travelling salesman problem," [Online]. Available: https://en.wikipedia.org/wiki/Travelling\_salesman\_problem. [Accessed 31 May 2018]. |
| [3] | B. Jain, "Write a program to print all permutations of a given string," GeeksforGeeks.org, [Online]. Available: https://www.geeksforgeeks.org/write-a-c-program-to-print-all-permutations-of-a-given-string/. [Accessed 1 June 2018]. |