A Basic Analysis of the Symmetric Traveling Salesman Problem Using the TSPLIB Repository

Gavin Wooley

October 2017

1. P, NP, and NPC.

The question of P $\stackrel{?}{=}$ NP is one of the most perplexing problems in computer science. It is deemed so challenging and worthy of completion that the Clay Mathematics Institute considers P $\stackrel{?}{=}$ NP as one of its Millenium Prize Problems–a list of unsolved problems in mathematics that reward one million dollars to a solution. Even considering tacking the question of P $\stackrel{?}{=}$ NP is outside the scope of this paper, but we note its importance in order to display the importance of the complexity classes of problems P, NP, and NPC.

It is informally stated that the class P consists of those problems that are solvable in polynomial time and that the class NP consists of those problems that are verifiable in polynomial time. Then, it is informally stated that NPC (read as NP-Complete) problems are those which are in NP and are as "hard" as any problem in NP. We will not formally define what it means to be "as 'hard' as any problem in NP," but we will look at one famous NPC problem–the symmetric traveling salesman problem.[1]

2. The Traveling Salesman Problem (TSP)

The TSP regards a problem within which a salesman must travel to n cities to sell his service or product, but he wishes to return to the city he began in and he wishes to visit each city only once exactly. We model the problem by saying that he wishes to traverse a tour around n cities, visiting each city once, and returning from where he began. Hence, a given solution is a tour or graph connecting those cities.

We will give the TSP formally in words as such:

> Given a weighted complete graph of n nodes, find a roundtrip of minimal total length visiting each node exactly once. The distance from node i to node j is the same as from node j to node i (that is, the graph of nodes is symmetric).

We will assume the TSP to belong to NPC by definition. Then, we may quickly begin work on a naive approach to solving the TSP by brute-force search. We can easily obtain the solution to the TSP by generating and searching among all permutations of solutions whilst keeping track of the solution with the lowest total length to traverse. Formally, we can define Brute-Force Search as follows, where C is a weighted complete graph of nodes (cities),

Brute-Force Search(C)

Let T be a list of all permutations of traversals in C such that each node is traversed to once exactly except for the first node, which is the same as the last node in the permutation.

$t \longleftarrow T[0]$

$i \longleftarrow 1$

while $i \leq T$.length

[1] Cormen, Leiserson, Rivest, Stein. *Introduction to Algorithms*

if $T[i]$.distance $< t$.distance

$t \longleftarrow T[i]$

$i = i + 1$

return t.

Then, we can easily note that Brute-Force Search $= O(p)$, where p is the number of permutations, as it will continue to iterate over every permutation of the graph. We can derive p in terms of n, where n is the number of nodes in the graph. Using the knowledge that permutations can be expressed as $m!$ where m is the number of items in the set, we can express p as $n!$. Hence, Brute-Force Search $= O(p) = O(n!)$.
It is easy to see that this is slow; $n!$ becomes larger than any polynomial as n becomes sufficiently large.