Notes - Algorithms & Data Structures - NP-Complete Problems

- Dr Nick Hayward

A brief intro to NP-Complete problems and associated algorithms.

Contents

Intro

In the *set-covering* problem, we need to calculate each possible set, regardless of the number of sets.

A common feature of *NP-complete* problems is the lack of a fast, exact algorithmic solution as the scale of the problem increases.

The classic example for *NP-complete* problems is, of course, the *Traveling Salesman* problem.

Traveling salesman

A salesman needs to visit a series of cities, initially starting out from Cairo.

The salesman, of course, would like to visit these cities using the shortest practical route.

So, to be able to calculate the shortest route, we need to initially calculate each and every possible route.

If we consider a trip that needs to visit five cities, how many routes do we actually need to calculate?

two cities

If we begin with a simple calculation, initially only two cities in the trip, we can quickly calculate two possible routes the salesman may choose for this trip.

![Traveling Salesman - 2 cities](np-traveling-salesman1.jpg

If we consider these routes, we might initially question why there is a duplication. Aren't these routes the same?

The inherent problem is that we cannot be certain that each route is the same distance, time, path, &c. Many routes will have one-way streets, perhaps only heading north, or diversions due to planning requirements, and so on. Different highways will also have different access ramps depending upon direction of travel.

In effect, these need to be recorded as two separate routes.

The other common query is whether we need to ensure we begin at a given city in the network.

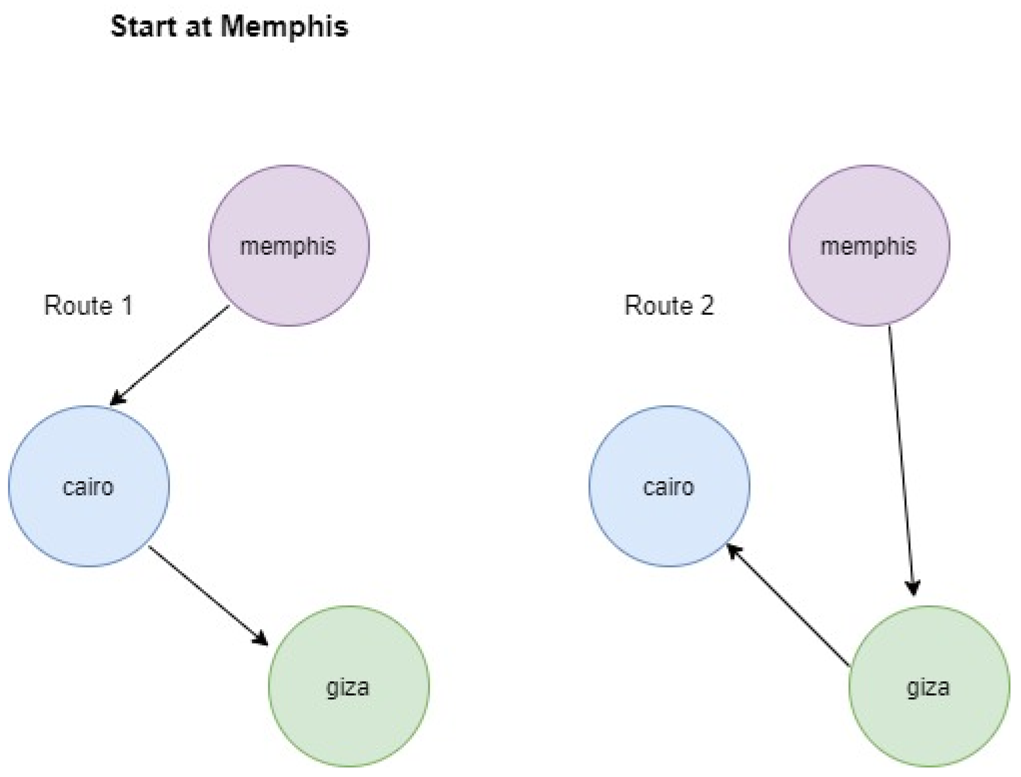
Whilst the above example begins in Cairo, we cannot assume this will always be true for each salesman, and every trip. The salesman, for example, may need to begin in Cairo, Giza, Memphis, and so on. There may be a delay in travel, and we need to restart at a different city, and so on. Again, an assumption we cannot hold as true.

So, the start location is unknown. The algorithm needs to be able to compute the optimal path for the salesman regardless of origin.

three cities

If we then add a third city to the current trip, we need to revise our calculation to consider the number of possible routes.

If we start at *Memphis*, we have two cities to visit, including *Cairo* and *Giza*.



So, with a starting point at Memphis, there will be two possible routes to Cairo and Giza.

A similar pattern may be seen if we begin at either Cairo or Giza, again returning two possible routes for each starting position.

So, for three cities we have six possible routes.

four cities

Then, if we add a fourth city to the trip, we may continue the calculation for possible routes.

So, we may add *Saqqara* as a city the salesman needs to visit during this trip.

If we start the trip at this new city, *Saqqara*, there are six possible routes.

We can quickly see a pattern emerging, which will define six available routes per available starting point.

So, with four possible start cities, six possible routes for each start, we get a simply calculation of `4 x 6 = 24` possible routes.

Of course, each time we add a new city we're increasing the number of routes we need to calculate for the trip.

add more cities

If we add more cities, we quickly start to see how the possible number of routes will grow rapidly.

For example,

no. of cities	possible routes
1	1 route
2	2 start cities x 1 route for each start = 2 total routes
3	3 start cities x 2 routes = 6 total routes
4	4 start cities x 6 routes = 24 total routes
5	5 start cities x 24 routes = 120 total routes
6	6 start cities x 120 routes = 720 total routes
7	7 start cities x 720 routes = 5040 total routes
8	8 start cities x 5040 routes = 40320 total routes
...	...

So, we can see a clear pattern to the growth of possible routes relative to the defined number of start cities.

This is known as the **factorial function**. For example, we can see that `5! = 120`.

If we check the total number of possible routes for 10 cities, we can calculate a total as `10!`, which equals `3,628,800`.

For just 10 cities in a route, we now need to calculate over three million possible routes.

We can see, of course, that the number of possible routes become very large, very quickly as the calculation executes.

This is why it is currently not feasible to compute a *correct* solution for this problem if there is a high number of cities in the trip.