Graph Theory, Applications and Combinatorics
UE19CS323 - Assignment 3
Summary on -
Dynamic Programming Treatment of the Travelling Salesman Problem

Author: Richard Bellman, Dated: Jan 1962, #Citations: 872

Date: 5/12/2021

Sumukh Raju Bhat	PES1UG19CS519

Link to the original paper:
Dynamic Programming Treatment of the Travelling Salesman Problem

Introduction:

Travelling salesman problem or simply TSP, is one of the most famous NP-hard and NP, and naturally a NP-complete problem known till date in the field of computer science and complexity theory in general. It is the key, like other NP-complete problems, directly or indirectly in solving many optimization problems. To name a few: vehicle routing problems, logistics, planning and scheduling.
Dynamic programming on the other hand is one of the most famous techniques to bring down the complexity of algorithms from exponential to polynomial. To name a few algorithms, 0/1 knapsack, subset sum, matrix chain multiplication etc. It is used where we have problems, which can be divided into similar sub-problems, so that their results can be reused. Mostly, these algorithms are used for optimization. Before solving the in-hand problem, dynamic programming will try to examine the results of the previously solved subproblems.
In the above mentioned paper, there is an attempt to apply dynamic programming technique on the travelling salesman problem to bring down the exponential time complexity to polynomial runtime or atleast improve the efficiency for a subset of inputs.
As it is infamously known, the problem TSP is stated as follows(As in the paper):
“A salesman is required to visit once and only once each of n different cities starting from a base city, and returning to this city. What path minimizes the total distance travelled by the salesman?” In technical

terms, TSP builds on the hamilton circuit problem and is concerned with computing the lowest cost hamiltonian cycle on a weighted (di)graph. It is to be noted that HCP or hamilton cycle problem, in itself is a NP-hard problem which can be proved by polynomially reducing an already known NP-complete problem called 3-SAT to HCP. In addition, we have to take care of the optimality condition to successfully solve TSP.

Previous works(At the time, the paper was published)/
Literature review:
The problem was treated as a combination of ingenuity and linear programming, which although produced results for simple inputs, say 4 cities, but failed when the number of cities was increased to 10. The current paper however, gives results and with good runtime for 17 cities. For larger numbers, the method presented, combined with various simple manipulations, may be used to obtain quick approximate solutions. (These methods were later published by 2 independent authors)

DP formulation/
Proposed Methodology:
Let f(i; j1, j2, …, jk) define the length of a path of minimum length from i to 0 which passes once and only once through each of the remaining k unvisited cities j1, j2, j3, …jk.
The whole TSP is broken down into a multistage decision problem. Let's assume that the tour begins at some origin or source, say city number 0. Fixing this arbitrarily won't mean we lose generality as the tour is the round trip as explained by the author.
For the tour to be optimal, the path from i through j1, j2, j3,...jk in some order and then to 0 must be minimum. Why? Let's assume that the above sentence is false. Then that would mean that we could find a shorter path and reduced shorter path from i through j1, j2, …, jk which would not be desirable by the TSP statement. Thus the entire TSP has been reduced to obtaining f(0; j1, j2, …jn) and a path which has this length.
Thus the recursive relation would be:
f(i; j1, j2, j3,...jk) = min {dijm + f(i; j1, j2, …, jm-1, jm+1, …. jk)} —--- (1)
where,

1 <= m <= k and dij is defined as the distance between ith and jth cities. We can compute the whole thing iteratively using the following steps:
S1. Initialization: f(i; j) = dij + dj0
S2. Loop Invariant: Use equation (1) to get f(i;j1, j2) from initialization, which inturn is used to get f(i; j1, j2, j3) and so on…
S3: Stopping condition: When f(0; j1, j2, …, jn) is obtained.
The informal proof that the above algorithm works is that the sequence of values of different values of m minimizes the expression in the braces on the right-hand side of equation (1) and hence gives a desired minimum path. And also the fact that the right hand of the equation (1) and also the stopping condition of the algorithm has j1, j2, j3, …, jn has all unique ji's and 1<=i<=n, means that all cities are visited only once.

Results:
Solving TSP using the good-old bruteforce way takes O(n!) time complexity. The mentioned algorithm, although doesn't convert it to a polynomially solvable problem as per as it's reputation, but still solves the problem in the order of O((2^n)*(n^2)) which is definitely an improvement as factorial would have grown faster than the exponential execution time function.
Space complexity on the other hand takes a serious hit by using this algorithm as an approach to solve TSP as we increase our tour size. The reason being that with the extent of memory and memory access speed available on those days, exponential memory requirement is too much to ask even for the best computer of those times. In the algorithm described, it is not necessary to store the order in which the cities are visited but we need to store the function f's results. Hence we need to tabulate all ways of choosing k quantities from n-1 quantities. The quantity is largest when k is the largest integer nearer to (n-1)/2. Considering the case when n-1 is even, using the Stirling's formula we get space complexity of the order of O(2^n).
The author has stated that for 11 cities the program can be run routinely, for 17 cities he requires the largest of the then current fast memory computers, but for 21 cities the algorithm is not feasible to be executed and the technology then is a few years behind.

Conclusions:

The author mentions that the memory limitations can be avoided by using some clever but complicated programming tricks by leveraging the fact that the distances will be integers and that we need not use all digits of one word to specify a distance. He also mentions that, in particular cases, one can easily reduce the number of cities by grouping subtours as a new distance in itself. As mentioned by the author, one of the most important advantages of using this algorithm is that we can easily apply many of the real life constraints that will occur in real life when compared to other solutions and approximate solutions to TSP. For instance, say we need the top 15 nearest neighbours of a given city i.

References in the paper:

1. On a linear programming combinatorial approach to the travelling salesman problem by G.B. Dantzig et. al.
2. Integer programming formulation and travelling salesman problems by C.E. Miller et. al.
3. Dynamic Programming by R. Bellman