Dynamic Programming

Travelling Salesman Problem

Travelling Salesperson Problem

- Given a directed graph G=(V,E) with edge costs cij. The Travelling salesman problem is to find a tour of minimum cost
- A tour is a directed simple cycle that includes every vertex in V
- Cost of a tour is the sum of cost of all the edges in the tour
- The edge cost is defined as below:

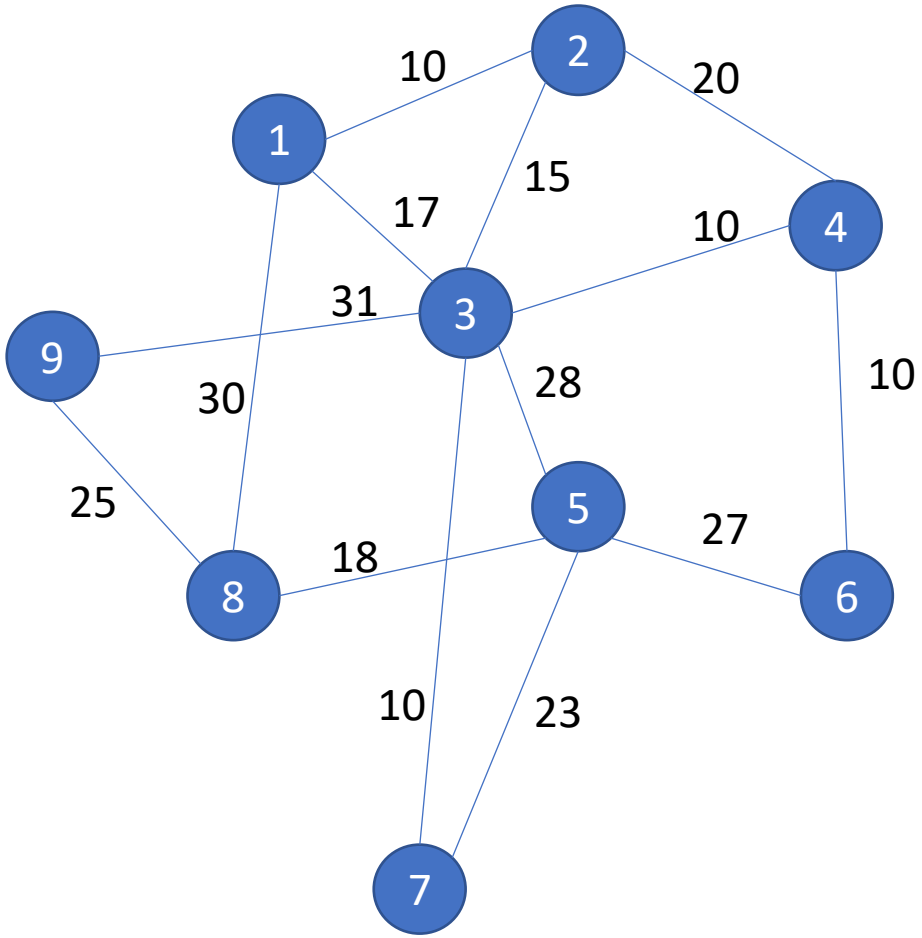
$$c_{ij} = \begin{cases} 0 & if\ i = j \\ > 0 & if < i,j > \in E \\ \infty & if < i,j > \notin E \end{cases}$$

Example Applications

- Suppose a postal van is to be routed such that it picks mails from mail boxes located at n different sites and return to the head office. An n+1 vertices graph can be used to represent the situation and optimal tour cost can be found
- A production environment where several commodities are manufactured on the same set of machines and the manufacturing proceeds in a cycle. Suppose in each cycle, n different commodities are produced and when the machines are changed from production of commodity i to commodity j, a change over cost cij is incurred. The task is to find an optimal order of manufacturing that minimizes the change over cost. It also involves cost of cycle change which is nothing but the change over cost from last commodity to the first commodity

Brute Force Approach

- We notice that for any n-vertices graph we can have n! ways of moving from one vertex travelling through all other vertices and returning to the starting vertex
- Thus, generating all possible sequence of vertices will require O(n!) time
- However, we can recursively define an optimal cost solution and apply dynamic programming to reduce the time complexity



(1,1) (1,2) (1,3) (1,4)…(2,2) (2, 3)…

(2,4,3)/ (2,3,4)/(3,4,2)/(3,2,4).. Min cost

(2,3,4,5) => (2,3,4) + (4,5)

(2,4,3) + (3,5)

(3,4,2) + (2,5)

} Min. cost

Defining a Solution

- Without loss of generality, we assume that an optimal tour is a simple path that begins and ends at vertex 1
- Every tour will consist of an edge <1,k> for some $k \in V - \{1\}$ and every vertex in $V - \{1, k\}$ exactly once
- We want to find a minimum cost tour and therefore the selected sequence must have minimum cost
- Let us define a function g{i,S} to be the length of a shortest path starting at vertex i, going through all vertices in S and terminating at vertex 1
- Then, we can write the length of the optimal tour as g(1, V-{1})

- We can thus write g(1, V-{1}) as:

$$g(1, V - \{1\}) = \min_{2 \le k \le n} \{c_{1,k} + g(k, V - \{1, k\})\}$$

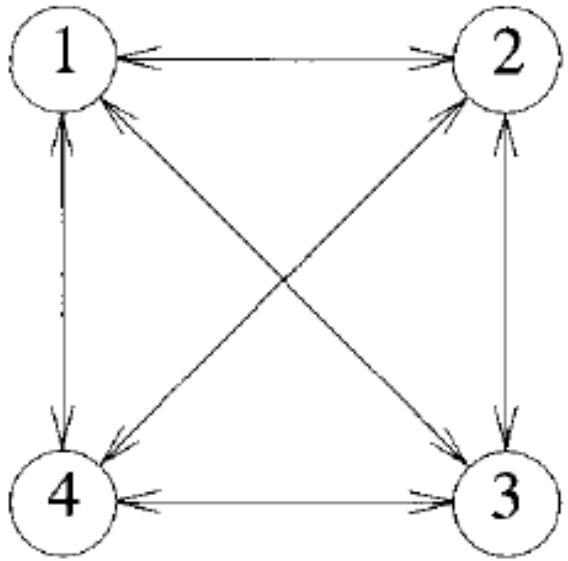
- We notice that in order to solve the above equation optimally, we must solve g(k, V-{1,k}) optimally
- This is because if it were not true then we could find a minimum cost tour with same value of $c_{1,k}$ resulting into an even lower cost tour for $g(1, V - \{1\})$ hence, contradicting our assumption that it gives minimum cost tour
- Thus, this problem exhibits **optimal substructure**
- Further, each $g(k, V - \{1, k\})$ has many subproblems in common for different values of k and this holds for any subproblem generated within Thus, this problem has many **overlapping subproblems**

Computing the Value of Optimal Solution

- We can calculate the value of an optimal solution by following a bottom-up approach
- We start by calculating the value of an optimal tour for S=0 and move upward to compute the optimal tour by adding an additional vertex in each pass
- A top-down approach having recursive calling with memoization can also be used to solve the problem
- Here, we discuss the bottom up solution by generalizing the previous equation as below:

$$g(i, S) = \min_{j \in S} \{c_{ij} + g(j, S - \{j\})\}$$

We can calculate the $g(1, V - \{1\})$ if we know the value of $g(k, V - \{1, k\})$ for all values of k



(a)

$$\begin{bmatrix} 0 & 10 & 15 & 20 \\ 5 & 0 & 9 & 10 \\ 6 & 13 & 0 & 12 \\ 8 & 8 & 9 & 0 \end{bmatrix}$$

(b)

Solution

- For all values i, we have $g(i, \phi) = c_{i1}$, $1 \leq i \leq n$
- We can use the above values to calculate the g(i,S) value for S=1 then S=2 and so on

$$g(2, \phi) = c_{21} = 5, g(3, \phi) = c_{31} = 6, \text{ and } g(4, \phi) = c_{41} = 8.$$

$$\begin{array}{lll} g(2,\{3\}) = c_{23} + g(3,\phi) = 15 & & g(2,\{4\}) = 18 \\ g(3,\{2\}) = 18 & & g(3,\{4\}) = 20 \\ g(4,\{2\}) = 13 & & g(4,\{3\}) = 15 \end{array}$$

$$
\begin{aligned}
g(2,\{3,4\}) &= \min\{c_{23}+g(3,\{4\}), c_{24}+g(4,\{3\})\} &= 25\\
g(3,\{2,4\}) &= \min\{c_{32}+g(2,\{4\}), c_{34}+g(4,\{2\})\} &= 25\\
g(4,\{2,3\}) &= \min\{c_{42}+g(2,\{3\}), c_{43}+g(3,\{2\})\} &= 23
\end{aligned}
$$

$$
\begin{aligned}
g(1,\{2,3,4\}) &= \min\{c_{12}+g(2,\{3,4\}), c_{13}+g(3,\{2,4\}), c_{14}+g(4,\{2,3\})\}\\
&= \min\{35,40,43\}\\
&= 35
\end{aligned}
$$

Constructing an Optimal Tour

- For constructing the optimal cost tour, we need to store the value of j that minimizes the cost value for each g(i,S)
- This j value can be traced in reverse order in order to find the optimal cost tour. Suppose J(,S) be this value then we can trace the solution for above problem as
- J(1,{2,3,4}) = 2 => J(2,{3,4}) = 4 => J(4,{3}) = 3
- Thus, optimal tour is 1,2,4,3,1

Complexity Analysis

$$\sum_{k=1}^{n} k \binom{n}{k} = n.2^{n-1}$$

- Let N be the number of g(i,S) that have to be computed before g(1, V-{1}) can be computed
- For each value of |S| there are n-1 choices for i
- The number of distinct sets S of size k not including 1 and i $\binom{n-2}{k}$

$$N = \sum_{k=0}^{n-2} (n-1) \binom{n-2}{k} = (n-1)2^{n-2}$$

- For each k, (k-1) comparisons will also be required for finding the minimum cost
- The algorithm will require $\theta(n^2 2^n)$ time for computation of optimal tour
- A serious drawback of the algorithm is the high space complexity $O(n2^n)$. This is too large even for modest values of n