CS301 Assignment 3

Dilara Nur Tekinoğlu

May 2021

a. Recursive formulation of the tourist problem: Identify the subproblems, observe the optimal substructure property and the overlapping computations, and then define the problem recursively respecting a topological ordering.

We can use Bellman-Ford algorithm with a modification to solve this tourist problem. The algorithm computes the shortest paths from a single source to every other vertex in the graph.

i. Subproblems: The shortest path from source vertex s to every other vertex v that uses i or fewer edges, if such a path exists. $d[i][v]$ indicates the shortest part from s to v that uses at most i edges.

ii. Optimal Substructure Property: By this property, optimal solution to the original problem is composed of optimal solutions to sub-problems. For example; if a city c is on the shortest path from s to v, then:

$$d(s, v) = d(s, c) + d(c, v).$$

This holds for every vertex on the shortest path from s to v (and for any v). If it does not, there would be a shorter path to c from s and c would be on the shortest path from s to v. This is clearly not possible because in such a case, we could take the shorter path to c and take $d(c, v)$ which would give us a shorter $d(s, v)$. So, there would be a contradiction. Therefore, we have optimal substructure property for this algorithm.

iii. Overlapping subproblems: We have many overlapping subproblems in this recursive solution. For example, d(s, k) can

used to compute d(s, v) for any v where the shortest sv path contains k.

iv. Recursive formula: We will first modify the Bellman-Ford as follows: We will keep following information for all vertices: 1. Existence of train station 2. Existence of bus station 3. Current location of the tourist on the vertex 4. Distance between train and bus stations (if both exist)

d(i,v) = 0 if i= 0 and v= source
d(i,v) = ∞ if i = 0 and v ! = source
d(i,v) = min(d(i-1,v), min $_{u,v \epsilon E}(d(i-1,u)+c(u,v))$ otherwise

where c(u, v) is computed as follows:
If the tourist is not at the desired station for journey (if the edge will be calculated for train but tourist is at bus station and vice versa) then, c(u,v) = w(u,v) + switchStationTime. Else, c(u,v) = w(u,v) where w(u,v) is the time the journey takes for current method (edge weight).

(b) Pseudocode of a naive recursive algorithm based on your recursive formulation, and its complexity analysis (i.e., the asymptotic time and space complexity).

```
def findShortestPaths(i, v):
    if i == 0 and v == source:
        return 0
    else if i == 0 and v != source:
        return ∞
    else
        return min(findShortestPaths(i-1,v), min (findShortestPaths(i-
1,u) + c(u, v) for all (u,v) in edges E)
```

Complexity Analysis: This naive recursive solution computes every possible path on the graph. Since there are exponential amount of sub-problems to be solved, asymptotic time complexity is exponential. Space complexity is linear.

(c) Pseudocode of an algorithm designed using dynamic programming (i.e., a recursive algorithm that builds solutions to your recurrence from top down with memoization, or an iterative algorithm that builds solutions to your recurrence from bottom up), and its complexity analysis (i.e., the asymptotic time and space complexity).

```
def findShortestPath(source):
    initialize distance array d
    d[source] = 0
    d[v] = ∞ for v in V except source
    for i =1 to n -1:
        for each edge (u,v):
            if d[v] > d[u] + c(u,v):
                then d[v] = d[u] + c(u, v)
```

This is an iterative algorithm that builds the solution from bottom up by filling an array.
c(u,v) will be calculated as explained above.
Complexity Analysis: This dynamic programming approach allows us to avoid duplicate computations. We compute the subproblems only once. Since there are V*E sub problems, total time complexity is $\theta(V.E)$. We use an additional array. Space complexity is $\theta(V)$.