CS345 Assignment-2

Ayush Shakya 180178
Soham Ghosal 180771

October 01, 2021

1 Aim

In this problem, we have been given a sequence of graphs $G[0]$,$G[1]$..$G[b]$, where each of these graphs $G[i]$ is connected and is defined as the tuple (V, E_i). Our aim is to come up with a polynomial time algorithm to find a sequence of paths $res[0]$,$res[1]$..$res[b]$ with the minimum cost keeping in mind the length of paths and the frequency of changes in the path structure.

For some constant $K > 0$, we define the cost of optimal path sequence $res[0], res[1], res[2], ...res[b]$ to be:

$$\text{cost}(res[0],res[1],...res[b]) = \sum_{i=0}^{b} len(res[i]) + K * changes(res[0], res[1], res[2], ...res[b])$$

Here we define the $l(res[i])$ to be the number of edges in the optimal path $res[i]$ and $changes(res[0], res[1], res[2], ...res[b])$ to be the number of indices i such that $0 \leq i \leq b - 1$, where $res[i] \neq res[i + 1]$ (i.e. atleast one element in both the arrays differs).

2 Sketch of Algorithm

2.1 Intuition

In order to reconstruct the sequence of paths with the minimum cost, we first need to determine the minimum cost of the optimal sequence of paths. An argument as to why greedy is not the right approach here, can be: At each step the cost function depends on both K as well as the sum of $l(res[i])$. If K is some large constant, then the product dominates and if K is some small constant, then the summation term dominates. Hence, we can never break it into a smaller subproblem of some predetermined size in terms of b. We need to consider the costs of all subproblems, before making a decision as to which index should be last changed to minimise the cost.

2.2 Notations

- $G_{intersection}$: For a particular run of i (line 6) and j (line 8), $G_{intersection}$ represents $(V, E_i \cap E_{i+1} ... \cap E_j)$
- $P[i][j]$: represents the shortest path between s and t in the graph $G_{intersection}$ upon BFS traversal
- $l[i][j]$: represents the length of shortest path between s and t in $G_{intersection}$, i.e. $l[i][j] = |P[i][j]|$ -1
- $costDP[i]$: to hold minimum cost corresponding to the optimal paths obtained from G[0...i]
- $diffDP[i]$: to hold the rightmost index on G[0...i] which differs from the previous optimal path.
- res : array to hold the optimal(of min. cost) sequence of paths, while considering G[0..b]

3 Pseudocode

Algorithm 1 Algorithm to compute a sequence of paths with minimum cost

```
procedure minCostPaths($G, K, s, t$)                    ▷ G is an array of all b graphs
    $b \leftarrow size(G)$
    for $0 \leq i \leq b$ do
        for every vertex v $\in G[i]$ do
            sort($G[i][v]$)
    for $0 \leq i \leq b$ do
        $G_{intersection} \leftarrow G[i]$
        for $i \leq j \leq b$ do
            $G_{intersection} \leftarrow G_{intersection} \cap G[j]$
            $(l[i][j], P[i][j]) \leftarrow$ BFS($G_{intersection}, s, t$)
    for $0 \leq i \leq b$ do
        $costDP[i] \leftarrow l[0][i] * (i+1)$
        $diffDP[i] \leftarrow -\infty$
        for $0 \leq j \leq i-1$ do
            $currentCost \leftarrow costDP[j] + l[j+1][i] * (i-j) + K$
            if $currentCost < costDP[i]$ then
                $costDP[i] \leftarrow currentCost$
                $diffDP[i] \leftarrow j$
    $v \leftarrow b$
    while $True$ do                                     ▷ Reconstructing the optimal sequence of paths
        $temp \leftarrow diffDP[v]$
        if $temp \neq -\infty$ then
            for $temp \leq i \leq v$ do
                $res[i] \leftarrow P[temp][v]$
            $v = temp - 1$
        else
            for $0 \leq i \leq v$ do
                $res[i] \leftarrow P[0][v]$
            $break$
    return $res$
```

4 Proof of Correctness

4.1 Recurrence Relation

Base Case : Consider the case when b=0. In this case, we only have 1 graph, ie. $G[0]$. Since $G[0]$ is connected (given), there definitely exists atleast one s-t path in $G[0]$. The shortest s-t path in $G[0]$ will be obtained in $P[0][0]$, and its length will be $l[0][0]$. Now inside the for loop (line 11), $costDP[0]$ will be assigned $l[0][0] * 1$ and $diffDP[0]$ will be assigned $-\infty$ which implies no change in the path, as expected. The res array finally stores the same path which is obtained in $P[0][0]$, which is also the expected optimal path, since we only have 1 graph.

Recurrence : Since $G[i]$ is connected $\forall$ i $\in$ [0..b], we can be sure that there exists at least one s-t path in every $G[i]$. Now, for some arbitrary i $\in$ [0..b], if there is no change in the expected paths $res[0], res[1], ...res[b]$ then the expected result would be $cost(res[0], res[1], ...res[b]) = (i+1) * l[0][0] + K * (0)$. The factor of $K * 0$ comes in because we have 0 changes among $res[0], res[1], ...res[b]$. Now, our algorithm calculates the currentCost variable to be $(j+1) * l[0][j] + l[j+1][i] * (i-j) + K$ (since $cost[j] = (j+1) * l[0][j]$ according to our induction hypothesis). This means that currentCost will always be greater than $costDP[i]$, hence our algorithm saves the result just like the base case, i.e., no change and all res[i] would have the optimal paths, stored in $P[0][0]$ (all the $P[i][j]$ would be same as $P[0][0]$, since no change ever occurred).

Let's now consider the case where **not** all expected paths $res[0], res[1], ...res[i]$ are the same, i.e. atleast one change occurs in the sequence of optimal paths. According to our notation, $diffDP[i]$ represents the rightmost index on $G[0..i]$ where there is a change in the sequence of optimal paths. Now, optimal cost must be :

$$cost(res[0], res[1], ...res[b]) = cost(res[0], ..., res[diffDP[i]]) + l[diffDP[i] + 1][i] * (i - diffDP[i]) + K$$

Here, $cost(res[0], ..., res[diffDP[i]])$ takes care of all the changes before i. To account for that one change in the path between $G[diffDP[i]]$ and $G[diffDP[i] + 1]$, we add a K. Finally, since no more changes occur from $diffDP[i] + 1$ to i, their length costs are simply summed up. In order to come up with this cost, we need to find $diffDP[i]$, which minimizes the expected cost. To do this, we simply loop over all possible values of $diffDP[i]$, i.e., 0 to i-1 to consider all possible values of costs choosing minimum among them.

The final recurrence, combining both the above cases can be written as:

$$\boxed{costDP[i] = min\big((i + 1) * l[0][i], \min_{0 \le j \le i-1}(costDP[j] + ((i - j) * l[j + 1][i]) + K)\big)}$$

4.2 Substructure optimality proof

The number of changes in the optimal solution is the number of subsections we have to break our problem into, where each subsection has the same path (hence a simple result). To get these subections, we break our problem into a subsection (from $diffDP[i]+1$ to i) and a smaller subproblem (mincost of $res[0...diffDP[i]]$). The subsection will trivially give an optimal solution, and we are left to calculate the optimal smaller subproblem which is similar to our original problem. Since we are calculating DP in a bottom-up fashion, if we claim that smaller subproblem gives an optimal solution, and our subsection (from $diffDP[i] + 1$ to i) also gives optimal results(trivially), then we can claim that their sum gives us the optimal solution to our original (bigger) problem. It should be noted that the subproblem and subsection are independent, which is crucial in our claim of optimality. Base case for this situation would be when we hit no change, which reduces the entire problem into a subsection whose optimal solution can be found trivially. Hence, we have shown how the original problem can be broken into simpler and calculable subproblem and subsection.

5 Time Complexity Analysis

We assume that the graphs have been given to us in the form of adjacency lists. We start by sorting the adjacency list of every vertex, in all of the graphs present in G array. To do this, we can use merge sort, which is based on divide and conquer paradigm. Assuming that m is max($|E_i|$) $\forall i \in [0...b]$ and n is $|V|$. The time complexity of this step would be $O(b * m * logn)$. The next step involves computing $G_{intersection}$ where we are using two nested for loops, each of size (b+1). Inside these loops, we perform intersection and BFS operations. Since we are using a sorted list structure, intersection would take O(m+n). The BFS function returns the length and path in a given graph from s to t, which takes O(m+n) time. Thus, the time complexity of this step is : $O(b^2 * (m + n))$.

$$\boxed{\text{Time Complexity of Preprocessing : } O(b^2 * (m + n) + b * m * logn)}$$

Then, we calculate $costDP[i]$ and $diffDP[i]$ $\forall i \in [0..b]$. For each i, we run a loop from [1..i), in which we perform only $O(1)$ operations (assignment and comparison, O(1) each). Thus, the time complexity of this step is $O(b^2)$. To retrieve the paths from diffDP array, we run a while loop whose number of iterations are bounded by O(b). Inside the loop, we make assignments to res in a for loop. Such assignments are again bounded by O(b). Thus, the time complexity of this while subsection is bounded by $O(b^2)$. Note that this is a loose upper bound.

$$\boxed{\text{Time Complexity of Processing : } O(b^2)}$$