CS345

Design and Analysis of Algorithms

Indian Institute of Technology, Kanpur

Tarun Kanodia (190902), Kaustubh Verma (190424)

Assignment 2

Date of Submission: 2th October, 2021

Question 1

Suppose we have a set of nodes V , and at a particular point in time there is a set E_0 of edges among the nodes. As the nodes move, the set of edges changes from E_0 to E_1, then to E_2, then to E_3, and so on, to an edge set E_b. For $i = 0, 1, 2, ..., b$, let G_i denote the graph (V, E_i). So if we were to watch the structure of the network on the nodes V as a "time lapse", it would look precisely like the sequence of graphs $G_0, G_1, ..., G_b$. We will assume that each of these graphs G_i is connected.

Now consider two particular nodes $s, t \in V$. For an $s-t$ path P in one of the graphs G_i , we define the length of P to be simply the number of edges in P, and we denote this by $\ell(P)$. Our goal is to produce a sequence of paths $P_0, ..., P_b$ so that for each i, P_i is an $s-t$ path in G_i . We want the paths to be relatively short. We also do not want there to be too many changes - points at which the identity of the path switches. Formally, we define $changes(P_0, ..., P_b)$ to be the number of indices i $(0 \leq i \leq b-1)$ for which $Pi \neq P_{i+1}$.

Fix a constant $K > 0$. We define the cost of sequence of paths $P_0, P_1 \dots P_b$ to be

$$cost(P_0, P_1, \dots, P_b) = \sum_{i=0}^{b} \ell(P_i) + K.changes(P_0, P_1, \dots, P_b)$$

Give a polynomial time algorithm to find a sequence of paths $P_0, ..., P_b$ of minimum cost.

Solution

Consider any sequence of paths $P_0, P_1, \ldots P_b$. It can be broken down into some contiguous blocks of sequence of paths such that each block has the same path (note that it is possible that each block has a length of 1,i.e, $P_i \neq P_{i+1} \ \forall i \in [0, b-1]$). Suppose for some i and j with $i < j$, all the paths from P_i to P_j are equal. This is only possible if the edges present in the path between vertices s and t exists in all the graphs $G_i, G_{i+1}, \ldots, G_j$. Thus we define a new graph, denoted as $G(i,j) = (V, \bigcap_{k=i}^{j} E_k)$,i.e, it has same vertex set as each of the graph, but has edges that are present in each of the graph from G_i to G_j. Thus if a path between vertices s and t exists in graph $G(i,j)$, it will exist in each of the graphs G_i to G_j. On the contrary, if the path does not exist in $G(i,j)$, it will not exist in at least one of the graphs, and for some k,$P_k \neq P_{k+1}$, so all the paths can not be same.

Let us define $\ell(i,j)$ be the length of the path between vertices s and t in $G(i,j)$, which can be found using **BFS** in $G(i,j)$ (since the length of the path is defined by the number of edges in the path). If no such path exists, then $\ell(i,j) = \infty$. We now present the following polynomial-time algorithm to compute the minimum cost and find the optimal sequence of paths. The notations in the algorithm used are as follows:

- $G(i,j)$: The intersection graph defined above,i.e, the one with vertex set V, and only the edges present in $G_i, \ldots, G_j$.
- $\ell(i,j)$: The length of $s-t$ path in $G(i,j)$. If no such path exists, $l(i,j) = \infty$.
- $P(i,j)$: The $s-t$ path in $G(i,j)$. If no such path exists, $P(i,j) = \phi$.
- $DP[i]$: Optimal cost when considering only the graphs from G_0 to G_i. So, the optimal cost for whole sequence shall be $DP[b]$.
- $last[i]$: The index j corresponding to the last change, i.e, in the optimal sequence of paths, $P_{j+1}, P_{j+2} \ldots P_i$ all are the same.

Algorithm

The pseudo code for the algorithm is given as follows:

Algorithm 1 Algorithm for finding optimal path sequence

procedure MIN-COST($G_0, G_1, \ldots, G_b, K, b, s, t$)
    for i from 0 to b **do** ▷ Precomputing all $G(i,j)$ and $\ell(i,j)$.
        $G(i,i) \leftarrow G_i$
        $\ell(i,i) \leftarrow$ path length by BFS on G_i from s
        for j from $i+1$ to b **do**
            $G(i,j) \leftarrow G(i,j-1) \cap G_j$
            Find $P(i,j)$ by **BFS** on $G(i,j)$ from s.
            if $P(i,j) \neq \phi$ **then**
                $\ell(i,j) \leftarrow$ length of $P(i,j)$
            else
                $\ell(i,j) \leftarrow \infty$.
            end if
        end for
    end for
    for $0 \leq i \leq b$ **do**
        $DP[i] \leftarrow (i+1).\ell(0,i)$ ▷ When $P_0, \ldots, P_i$ are same (if possible).
        $last[i] = -1$
        for $0 \leq j \leq i-1$ **do** ▷ Executes only when $i > 1$.
            if $DP[i] > DP[j] + (i-j).\ell(j+1,i) + K$ **then**
                $DP[i] \leftarrow DP[j] + (i-j).\ell(j+1,i) + K$
                $last[i] \leftarrow j$.
            end if
        end for
    end for
    $k \leftarrow b$ ▷ Finding the optimal sequence of paths
    $S[0..b] \leftarrow \phi$ ▷ Array storing optimal path sequence.
    while $k \neq -1$ **do**
        $S[last[k]+1, k] \leftarrow P(last[k]+1, k)$ ▷ Assigning a range to S
        $k \leftarrow last[k]$
    end while
    return $S, DP[b]$ ▷ Optimal Sequence and Minimum Cost returned
end procedure

Proof Of Correctness

Since this is a dynamic programming algorithm, we prove the correctness by proving the following:

- **Optimal Substructure Property:** Note that the given problem possesses the optimal substructure property. This can be seen as follows: Consider any sequence of paths $P_0, P_1, \dots P_b$. In the trivial case, if there are no changes in the sequence, then there are no subproblems and the cost is simply $\sum_{i=0}^{b} \ell(P_i) = (b+1).\ell(0,b)$. Let us assume that there is atleast one change in the sequence of paths. Let the index of the last change be j so that $P_{j+1}, \dots, P_b$ are same. In this case, $cost(P_0, P_1, \dots, P_b) = cost(P_0, P_1, \dots, P_j) + (b-j).\ell(j+1,b) + K$. Hence the problem is broken down into subproblems cost of which are directly added, and one of which is the trivial case having fixed cost.Also since $P_j \neq P_{j+1}$, the subproblems are independent, as $changes(.)$ only considers the adjacent paths in cost. So for the given sequence to be optimal, $cost(P_0, P_1, \dots, P_b)$ has to be minimum and for this, $cost(P_0, P_1, \dots, P_j)$ has to be minimum and hence the sequence $P_0, P_1, \dots P_j$ must be optimal. Since the optimal solution for the problem can be obtained from optimal solution of it's subproblems, the given problem possess the optimal substructure property.

- **Recurrence Relation:** Consider the sequence of paths $P_0, P_1, P_2, \dots, P_i$. There are two cases:

  1. **No change in paths:** In this case, all the paths $P_0, P_1, .., P_i$ are same, each being equal to $P(0,b)$ obtained by BFS on $G(0,b)$. Let $C(i)$ be the cost for a sequence of paths $P_0, P_1, .., P_i$. Then $C(i) = (i+1) * \ell(0,i)$.
  2. **At least one change in path:** In this case, let j be the index of last change, i.e, $P_j \neq P_{j+1}$ and $P_{j+1} = P_{j+2}.. = P_i = P(j+1,i)$. In this case, $C(i) = C(j) + (i-j) * \ell(j+1,i) + K$. K is added due to the change between P_j and P_{j+1}. So to get minimum among this case, we consider all possible indices from 0 to $i-1$ as j, and take minimum of cost associated with all of them.

  So to get the minimum cost for the sequence, we take the minimum of the two cases, and hence:

$$C(i) = \min((i+1)\ell(0,i), \min_{0 \le j \le i-1}(C(j) + (i-j).\ell(j+1,i) + K)) \tag{1.1}$$

Note that while calculating $C(i)$, we calculate $C(j)$ for all $j < i$, and while calculating $C(i+1)$, we calculate $C(j)$ for all $j \leq i$, most of which were already calculated while computing $C(i)$. Hence this problem has **overlapping subproblems**.

Now since the problem has both optimal substructure and overlapping subproblems, dynamic programming can be applied. This has been done in the algorithm above. Note the precomputation of $G(i, j)$ and $\ell(i, j)$ is done to improve the time complexity. ($G(i, j)$ can be simply calculated from $G(i, j-1)$ and G_j, if both values are stored). Moreover, $\ell(i, j)$ is calculated by doing a breadth first search on $G(i, j)$ from vertex s. Since G_i is connected $\forall\ i$, there always exists a path in each of G_i.

We conclude the proof by showing from induction that if all the costs $DP[0] \ldots, DP[i-1]$ have been calculated correctly, then $DP[i]$ is calculated correctly by the algorithm.

- **Base case:** When $i = 0$, $G(0, 0) = G_0$, and hence $P(0, 0) = P_0$ is the shortest path obtained by breadth first search on G_0. When $i = 0$, the inner loop does not execute and $DP[0] \leftarrow \ell(0, 0)$ which is the minimum cost. Moreover, $last[0] = -1$, and hence $S \leftarrow P(0, 0)$ which is the optimal sequence.

- **Inductive Step:** We assume that all the values $DP[0], DP[1], \ldots, DP[i-1]$ are calculated correctly. The path P_i can either be the shortest in G_i, or the shortest in $G(i-1, i)$ (when $P_i = P_{i-1}$),...,or the shortest in $G(0, i)$ (when all paths are same). Note that our algorithm's inner loop follows Equation 1.1, and hence finds the minimum by considering all the above cases. Hence $DP[i]$ calculated must be correct.

  Moreover, the correct sequence of path can be retrieved easily by storing the index of last change for the current index, as an indicator of the subproblem that lead to the optimal solution for the current index.

Time Complexity Analysis

Let $|V| = n$ and $\max_{0 \leq k \leq b} |E_k| = m$. Our algorithm consists of three steps:

- **Precomputation:** Note that for any of the $G(i, j)$, number of vertices = n and number of edges $\leq m$. Finding $G(i, j)$ from $G(i, j-1)$ and G_j takes $O(n+m)$ time since we have to iterate over all the vertices and edges in $G(i, j-1)$ and G_j (Presence of common edges can be found by a DFS on each of the graphs). Finding shortest path in $G(i, j)$ also takes $O(m + n)$ time, which is by BFS.

  So one iteration of inner loop takes $O(m + n)$ time. There are $(b + 1)^2$ such iterations, for each $0 \leq i, j \leq b$. So overall time complexity of precomputation is $O((m + n)(b + 1)^2) = O((m + n).b^2)$.

- **Computation of DP:** Each iteration of inner for loop takes $O(1)$ time. There are again $O(b^2)$ number of iterations, hence the complexity becomes $O(b^2)$.

- **Retrieving optimal path:** Any path in G_i has an $O(n)$ vertices, so copying P_i into $S[i]$ takes $O(n)$ time (if path is stored as a list). Ultimately each of the $P_0, P_1 \dots P_b$ are copied into $S[i]$, the complexity of this part is $O(bn)$.

Thus the overall complexity of the algorithm is $O((m+n).b^2) + O(b^2) + O(bn) = O((m+n).b^2)$, which is a polynomial time algorithm.