CS345: Algorithms - II
Assignment 2 Solutions

Aditi Goyal - 190057
Yatharth Goswami - 191178

October 2, 2021

I Problem 1 (Difficult) Solution

In this problem, we were asked to compute minimum cost paths dynamically over a changing network of nodes and had to optimise a particular type of cost metric. We solved this problem using the technique of Dynamic Programming and the solution will be explained below with proof of correctness. Let us define the problem formally first that we will be tackling.

Problem Statement: We are given a set of nodes V, and a set of edges which change dynamically over different timestamps. At timestamp i, we have the set of edges as E_i. Let us denote the graph at time $i \in \{0, 1 \dots b\}$ by $G_i = (V, E_i)$. Another assumption is that all the G_is are connected. The problem is to look at two particular nodes s and t and define a path connecting them in graph G_i as P_i. Given a constant K, we are asked to find a sequence of paths $P_0, P_1, \dots P_b$ that minimises

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

Let us define some notations first that we will be using throughout the rest of the solution.

1.1 Notations and definitions

- $DP[i][j]$: For $0 \le i, j \le b$ this represents the minimum cost of sequence of paths $P_0, P_1 \dots P_i$ with $changes(P_0, P_1 \dots P_i) \le j$ where P_l is a path from s to t in G_l for $0 \le l \le i$. It is also possible that no such sequence of path exists where at most j changes can be done, in which case $DP[i][j] = \infty$.

- E_{ij}: For $0 \le i \le j \le b$ it is defined as $E_{ij} = \bigcap_{i \le l \le j} E_l$.

- C_{ij} : For $0 \le i \le j \le b$, Minimum cost to reach from s to t in the graph $G = (V, E_{ij})$. It is also possible that t might not be reachable from s with edge set defined as E_{ij}, in which case $C_{ij} = \infty$. Note that it is equivalent to the minimum cost of a sequence of paths $(P_i, P_{i+1}, \dots, P_j)$ such that P_k is a path from s to t in the graph G_k and $changes(P_i, P_{i+1}, \dots, P_j) = 0$.

- For $0 \le i, j \le b$, $(P_0^{ij}, P_1^{ij}, \dots P_i^{ij})$ is a tuple that represents an optimal sequence of paths with at most j changes, where P_l^{ij} is a path from s to t in G_l for $0 \le l \le i$, for which the cost is $DP[i][j]$.

- For $0 \le i \le j \le b$, $(C_i^{ij}, C_{i+1}^{ij}, \dots C_j^{ij})$ is a tuple that represents an optimal sequence of paths (for the problem of minimising $cost(P_i, P_{i+1} \dots, P_j)$ with $changes(P_i, P_{i+1} \dots P_j) = 0$) for which the cost is C_{ij}.

- S^{ij} : For $0 \le i, j \le b$, it denotes the set of all sequence of paths $(P_0, P_1, \dots P_i)$ such that $changes(P_0, P_1 \dots P_i) \le j$.

1.2 Recursive formulation

We notice that the given problem has an optimal substructure which we can exploit to devise a recursive relation which can solve the problem in polynomial time complexity. For this let us prove the recursive relation as a claim below. We will be using the same set of notation which has been defined in the section 1.1.

Claim 1.2.1. $DP[i][j] = min(C_{0i},\ min_l\ (DP[l][j-1] + K + C_{(l+1)i}))$ *where* $0 \le l < i$

Proof. We prove this in two parts.

- $DP[i][j] \le min(C_{0i},\ min_l\ (DP[l][j-1] + K + C_{(l+1)i}))$ where $0 \le l < i$

  For proving this part, we will show that $DP[i][j] \le C_{0i}$ and $DP[i][j] \le min_l\ (DP[l][j-1] + K + C_{(l+1)i})$ where $0 \le l < i$. Let us first prove each of the two terms.

  1. $DP[i][j] \le C_{0i}$
     Proof: Notice that $DP[i][j]$ defines the optimal cost of set of paths from s to t in the prefix i of set of graphs with at most j changes. Also recall that C_{0i} defines the minimum cost to reach from s to t with no changes in the path. Therefore we can see that problem of finding C_{0i} (if exists) is a sub-problem of the problem addressed by $DP[i][j]$. We need not worry about the case when C_{0i} does not exist (because in that case it is defined to be ∞). Since, C_{0i} represents the cost of a sequence of paths which lies in set S^{ij} and $DP[i][j]$ is the minimum cost for all path sequences in S^{ij},
     $$\boxed{DP[i][j] \le C_{0i}} \tag{I1}$$
  2. $DP[i][j] \le min_l\ (DP[l][j-1] + K + C_{(l+1)i})$ where $0 \le l < i$
     Proof: As told before, $DP[i][j]$ defines the optimal cost of set of paths from s to t in the prefix i of set of graphs with at most j changes. For understanding the right hand side, there might be two cases arising, for any $l \in \{0, \dots i-1\}$ either the end of any sequence of paths corresponding to $DP[l][j-1]$ and any path corresponding to $C_{(l+1)i}$ will be same or they will be different. Note that if $DP[l][j-1]$ or $C_{(l+1)i}$ is not defined (∞), for some l, the inequality is trivially true for that l. So, we will not worry about these cases further in our study.

     Consider for any $l \in \{0, \dots i-1\}$ the concatenation of the paths ($P_0^{l(j-1)}$, $P_1^{l(j-1)}, \dots P_l^{l(j-1)}$) and the path ($C_{l+1}^{(l+1)i}, C_{l+2}^{(l+1)i}, \dots C_i^{(l+1)i}$) (say P). Note that these sequence of paths may not be unique, but the proof ahead does not rely on any specific path sequence so we can consider any optimal sequence of paths without loss of generality. In the concatenated sequence, there will be atmost j changes (atmost $j-1$ changes till l and atmost 1 change between l and $l+1$). So, this concatenated sequence of path is an element of S^{ij}.

     In case the paths $P_l^{l(j-1)}$ and $C_{l+1}^{(l+1)i}$ are different, this leads $DP[l][j-1] + K + C_{(l+1)j}$ to be the cost of a solution of problem of at most j changes in the prefix i, since $DP[l][j-1]$ corresponds to the cost of solution of at most $j-1$ changes in prefix l, adding K reflects the cost due to one change and adding $C_{(l+1)i}$ term reflects the minimum cost of choosing the same path in the suffix from $l+1$ to i. Since $DP[i][j]$ is the minimum cost for the sequence of paths in the set S^{ij} while P is just an element of S^{ij}, we get that $DP[i][j] \le DP[l][j-1] + K + C_{(l+1)i}$ for each such l.

     Now, for the other case consider any $l \in \{0, \dots i-1\}$, in case the two paths (the end of sequence of paths corresponding to $DP[l][j-1]$ ($P^{l(j-1)_l}$) and any path corresponding to $C_{(l+1)i}$) are same. Consider concatenating the paths from the solution of $DP[l][j-1]$ and the path from $C_{(l+1)i}$ (call it P). Since, the paths $P_l^{l(j-1)}$ and $C_{l+1}^{(l+1)i}$) are same, hence on concatenating the two sets of paths, $change(P) = change(P_0^{l(j-1)}, P_1^{l(j-1)}, \ldots P_l^{l(j-1)}) \leq j-1$. Hence P will have at most $j-1$ changes (which arise due to the solution from $DP[l][j-1]$). Therefore, it becomes easy to notice that this sequence of paths obtained is an element of the set S_{ij}. Since $DP[i][j]$ is the minimum cost for the sequence of paths in the set S^{ij} while P is just an element of S^{ij}, we get that $DP[i][j] \leq DP[l][j-1] + C_{(l+1)i}$ for any such l. Now, since $K > 0$ therefore we get that $DP[i][j] \leq DP[l][j-1] + C_{(l+1)i} < DP[l][j-1] + K + C_{(l+1)i}$.

Therefore we get that $\forall l \in \{0, \ldots i-1\}$, $DP[i][j] \leq (DP[l][j-1] + K + C_{(l+1)i})$ and therefore

$$\boxed{\text{DP[i][j]} \leq min_l \ (DP[l][j-1] + K + C_{(l+1)i}) \text{ where } 0 \leq l < i} \qquad \text{(I2)}$$

Now, combining the two results obtained in (I1) and (I2) above, we get that

$$\boxed{DP[i][j] \leq min(C_{0i}, \ min_l \ (DP[l][j-1] + K + C_{(l+1)i})) \text{ where } 0 \leq l < i} \qquad \text{(I3)}$$

- $DP[i][j] \geq min(C_{0i}, \ min_l \ (DP[l][j-1] + K + C_{(l+1)i}))$ where $0 \leq l < i$

Consider any optimal solution $(P_0^{ij}, P_1^{ij}, \ldots P_i^{ij})$ for $DP[i][j]$ (if exists). If there is no solution for $DP[i][j]$, the inequality is trivially true. There are two cases to consider:

1. $changes(P_0^{ij}, P_1^{ij} \ldots P_i^{ij}) = 0$: Here, $P_0^{ij} = P_1^{ij} = \ldots P_i^{ij}$. In this case, $DP[i][j] = (i+1) length(P_0^{ij})$ (by definition of cost of sequence of paths). Also, all the edges in P_0^{ij} must be present in E_{0i} because the same path is present in $E_0, E_1, \ldots E_i$. Thus, by definition of C_{0i},

$$C_{0i} \leq (i+1) length(P_0^{ij}) = DP[i][j] \qquad \text{(I4)}$$

   because C_{0i} considers the shortest path in E_{0i} and P_0^{ij} is one of the paths in E_{0i}.
2. $changes(P_0^{ij}, P_1^{ij} \ldots P_i^{ij}) \geq 1$: There exists $l' < i$ such that $P_{l'}^{ij} \neq P_{l'+1}^{ij}$ and $P_{l'+1}^{ij} = P_{l'+2}^{ij}, \cdots = P_i^{ij}$. Informally, we are considering the first point where path is different if we start from P_i^{ij} to P_0^{ij}. Since one change is exhausted at the place l' and $changes(P_0^{ij}, P_1^{ij} \ldots P_i^{ij}) \leq j$ by definition of $DP[i][j]$, we have that $changes(P_0^{ij}, P_1^{ij} \ldots P_{l'}^{ij}) \leq j-1$. Now since $DP[l'][j-1]$ considers all sequence of paths $P_0, P_1, \ldots P_{l'}$ where at most $j-1$ changes take place,

$$cost(P_0^{ij}, P_1^{ij} \ldots P_{l'}^{ij}) \geq DP[l'][j-1] \qquad \text{(I5)}$$

   Also,

$$DP[i][j] = cost(P_0^{ij}, P_1^{ij} \ldots P_{l'}^{ij}) + K + cost(P_{l'+1}^{ij}, P_{l'+2}^{ij}, \ldots P_i^{ij}) \qquad \text{(I6)}$$

   Substituting equation 6 in 5,

$$DP[i][j] - K - cost(P_{l'+1}^{ij}, P_{l'+2}^{ij}, \ldots P_i^{ij}) \geq DP[l'][j-1] \qquad \text{(I7)}$$

Since $P^{ij}_{l'+1} = P^{ij}_{l'+2}, \cdots = P^{ij}_{i}$, $C_{(l'+1)i}$ is finite, Also, all of P^{ij}_{i} must have all of its edges in $E_{(l'+1)i}$ and since P^{ij}_{i} just one of the paths in $E_{(l'+1)i}$,

$$C_{(l'+1)i} \leq (i-l')length(P^{ij}_{i}) = cost(P^{ij}_{l'+1}, P^{ij}_{l'+2}, \ldots P^{ij}_{i}) \tag{I8}$$

Substituting equation 8 in 7,

$$DP[i][j] - K - C_{(l'+1)i} \geq DP[l'][j-1] \tag{I9}$$

which implies $DP[i][j] \geq K + C_{(l'+1)i} + DP[l'][j-1]$. So we have proved that in this case, there exists l' such that $DP[i][j] \geq K + C_{(l'+1)i} + DP[l'][j-1]$. Since we take minimum over all l', the inequality $DP[i][j] \geq min_l\ (DP[l][j-1]\ +\ K\ +\ C_{(l+1)i})$ where $0 \leq l < i$ will hold.

Case 1 $\implies DP[i][j] \geq C_{0i}$
Case 2 $\implies DP[i][j] \geq\ min_l\ (DP[l][j-1]\ +\ K\ +\ C_{(l+1)i})$ where $0 \leq l < i$

Since we are taking minimum of $min_l\ (DP[l][j-1]\ +\ K\ +\ C_{(l+1)i})$ and C_{0i}, we have

$$\boxed{\text{DP[i][j]} \geq min_l\ (DP[l][j-1]\ +\ K\ +\ C_{(l+1)i}) \text{ where } 0 \leq l < i} \tag{I10}$$

Hence, combining equations 3 and 10, we get

$$\boxed{\text{DP[i][j] = min(C}_{0i},\ min_l\ (DP[l][j-1]\ +\ K\ +\ C_{(l+1)i})) \text{ where } 0 \leq l < i} \tag{I11}$$

Hence proved. This proof also provides an insight on how the algorithm can be designed. □

1.3 Overview

The algorithm computes $DP[i][j]$ for all $0 \leq i, j \leq b$ and it also stores which term in the expression $min(C_{0i},\ min_l\ (DP[l][j-1]\ +\ K\ +\ C_{(l+1)i}))$ where $0 \leq l < i$ corresponded to minimum which can be used to reconstruct the sequence of paths.

Using the definition of C_{ij} as defined above to be the minimum cost to reach from s to t in the graph $G = (V, E_{ij})$. For a single graph, the cost of a path is defined to be same as it's length. Therefore, we can equivalently turn the graph G into a weighted edge graph $G' = (V, E_{ij})$, with each edge having weight 1. So, this problem of finding C_{ij} can be thought of as finding the shortest distance path between s and t in the graph G'. Since the weights of the edges are non-negative, therefore we can just find the minimum cost using Dijkstra's algorithm.

Note: In the pseudo code provided below, we have used the procedure Dijkstra, which we have not provided the implementation for. We have used it as a blackbox which will run Dijkstra algorithm starting from a source node s and returns the minimum cost as well as minimum cost path to t as a solution.

1.4 Pseudo code for sequence of paths with minimum cost

Algorithm 1: Algorithm to find sequence of paths with minimum cost

Input: Set of vertices V, sequence of edge sets $E_0, E_1 \ldots E_b$, vertices s, t and K

Output: Sequence of paths $P_0, P_1, \ldots P_b$ with minimum cost

```
C ← 2-D array to store the costs C_ij defined in section 1.2
M ← 2-D array of paths to store any minimum path corresponding to C_ij
Parent ← 2-D array to store the index of first point of change from the last
  path in the sequence of paths corresponding to DP[i][j]
Function FindC(V, E_0, E_1, ... E_b, s, t):
    for i in 0 to b do
        E ← E_i
        for j in i to b do
            E ← E ∩ E_j
            (C_ij, M_ij) ← Dijkstra(V, E, s, t)
    return C, M
Function ComputeDP(V, E_0, E_1, ... E_b, s, t):
    C, M ← FindC(V, E_0, E_1, ... E_b, s, t)
    for i in 0 to b do
        DP[i][0] ← C_0i
        Parent[i][0] ← −1
    for j in 1 to b do
        for i in 0 to b do
            DP[i][j] ← C_0i
            Parent[i][j] ← −1
            for l in 0 to i − 1 do
                if DP[i][j] > (DP[l][j − 1] + K + C_(l+1)i) then
                    DP[i][j] ← (DP[l][j − 1] + K + C_(l+1)i)
                    Parent[i][j] ← l
    return DP, Parent, M
Function ComputePaths(V, E_0, E_1, ... E_b, s, t):
    DP, Parent, M ← ComputeDP(V, E_0, E_1, ... E_b, s, t)
    Paths ← GetRecursivePath(b, b, Parent, M)
    return Paths
Function GetRecursivePath(i, j, Parent, M):
    l ← Parent[i][j]
    P′ ← [M_(l+1)i || M_(l+1)i ... || M_(l+1)i] // Concatenating M_(l+1)i i − l times
    if l == −1 then
        return P′
    Paths ← GetRecursivePath(l, j − 1, Parent, M) || P′
    return Paths
0
```

1.5 Proof of correctness

- **Correctness of function FindC**: This function computes the value of C array. As told previously, C_{ij} stores the minimum cost as defined in Section 1.1. This provided function first computes E_{ij} as defined in the Section 1.1 and then applies Dijkstra on the graph formed by the edge set E_{ij} and vertex set V. Hence, by definition of C_{ij} this function returns the correct value of C_{ij} and M_{ij}. We will now prove that applying Dijkstra's algorithm on the graph $G = (V, E_{ij})$ solves the problem of computing a sequence of paths $(P_i, P_{i+1}, \ldots, P_j)$ such that P_k is a path from s to t in the graph G_k and $changes(P_i, P_{i+1}, \ldots, P_j) = 0$.

  Say, there exists another common path from s to t (say P) corresponding to C_{ij} with smaller cost than the one found by Dijkstra. Since, P exists in each of the graphs (G_i to G_j), hence it will also be present in $G = (V, E_{ij})$. But then, applying Dijkstra on G should have returned the path with cost same as P, which is a contradiction to the fact that cost of P is smaller than the one found by Dijkstra. Hence, applying Dijkstra Algorithm works in this case and hence the correctness of this function.

- **Correctness of function FindPaths:** Let the sequence of paths returned by **ComputePaths** be $(P_0, P_1, \ldots P_m)$. We have that $m = b$ and edges in P_l are subset of E_l for any l and it forms a path from s to t in G_l. Assume that in k^{th} recursive call, i takes value i_k and l takes value l_k. We have that in k^{th} call, $i_k - l_k$ paths are appended to the sequence of paths. Thus total number of paths = $i_1 - l_1 + i_2 - l_2 \ldots i_m - l_m$ where m is the number of recursive calls in which the algorithm terminates (proof given in time complexity analysis). Also, from line 30 and 34 in the code, we have $i_{k+1} = l_k$ for any $1 \leq k < m$. Thus, total number of paths = $i_1 - l_m = b + 1$ (initially i is b and the algorithm terminates when l is -1). Also, note that in k^{th} call, intersection of edge sets from E_{l_k} to $E_{(l_{k+1}-1)}$ generate P_{l_k} to $P_{(l_{k+1}-1)}$ each of which is a path from s to t. Therefore, we have that P_l contains edges from the graph G_l. Let us now prove the correctness of the rest of the algorithm with help of some claims.

  Claim 1.5.1. *The sequence given by $DP[b][b]$ is the required sequence*

  Proof. Consider any path $P_0, P_1, \ldots P_b$. $changes(P_0, P_1, \ldots P_b) \leq b$ (= b, if we change at all indices). Thus, the optimal path sequence corresponding to $DP[b][b]$, will be the best among all the possible path sequences of length b. □

  Claim 1.5.2. *The cost of the sequence of paths returned by the function $GetRecusivePath(i, j, Parent, M)$ is $DP[i][j]$.*

  Proof. We will prove this using induction on i (number of edge sets).
  Base case (i = 0): In this case, the value of parent will be -1 for all $j \in \{0, 1, \ldots b\}$, as can be seen in the $ComputeDP$ function. Hence, the algorithm will trivially return the shortest path present in the graph G_0. Hence the algorithm works for the case of $i = 0 \forall j \in \{0, 1, \ldots b\}$ (because 0 changes will be there when $i = 0$).
  Induction step: Assume that the function correctly returns the answer for all $i \leq k$ and $\forall\ j \in \{0, 1, \ldots, b\}$ and we will prove that it also returns correct sequence of paths for $i = k + 1$ and any j. For the case when $j = 0$, the

value of l becomes -1 and hence the function correctly returns the path M_{0i} concatenated $i+1$ times. For the other cases when $j \geq 1$, when calling the function $GetRecursivePath$ with $i = k+1$, it can either happen that value of l inside the function becomes -1 or something else. If $l == -1$, this means that inside the $ComputeDP$ function $parent[k+1][j]$ will not get updated in lines 20-23 and hence $DP[k+1][j]$ will also store the value $C_{0(k+1)}$, which is correctly returned by the algorithm using the procedure $findC$. In the other case, when $l \neq -1$, l will be less than $k+1$ and we are returning the optimal solution of $DP[l][j-1]$ concatenated with P'. Note that by induction hypothesis, the path returned by recursive call with $i = l$ and $j = j-1$ returns an optimal sequence of paths with cost $DP[l][j-1]$. We now need to prove that this sequence of path concatenated with P' has cost of $DP[k+1][j]$. There may be two cases arising, one that the final path returned from the recursive call (say, $\{P_0, P_1, \ldots, P_l\}$) and the sequence of paths P' (variable as defined in pseudo code) (say $\{P'_0, \ldots P'_{i-l-1}\}$) have the same paths P_l and P'_0 or different paths P_l and P'_0. Also by our DP formulation, we have that

$$DP[i][j] = DP[l][j-1] + K + C_{(l+1)i} \text{ where } l = Parent[i][j]$$

In the case of different paths, we are done since the cost of the returned path is same as $DP[l][j-1] + K + C_{(l+1)(k+1)}$, which is indeed $DP[k+1][j]$ (using our DP formulation). Now, we will prove that the other case is not possible, i.e. they cannot have same paths P_l and P'_0. Assume that this happened to be the case, then the path returned by the function will be a valid path (as proved above using induction hypothesis) in $S^{(k+1)j}$ and have the cost of $DP[l][j-1] + C_{(l+1)(k+1)}$ which is strictly less than $DP[l][j-1] + K + C_{(l+1)(k+1)} = DP[k+1][j]$, which cannot be true, since $DP[k+1][j]$ is the optimal cost of any sequence of paths in $S^{(k+1)j}$. We have not made any assumption on j, it holds $\forall j$. Hence proved the induction step. □

Using the above claim, function **GetRecursivePath** when called with $i = b$ and $j = b$, will return the optimal path with cost $DP[b][b]$ and hence by claim 1.5.1, the returned path will be the required path.

1.6 Time complexity analysis

- Lines 5,7 have one loop each, iterating over $\mathcal{O}(b)$ values.
- Line 8 can compute intersection of edge sets in $\mathcal{O}(n^4)$ (checking for each edge in first edge set and iterating over all edges in second to see if it lies there too).
- Dijkstra can be done in $\mathcal{O}(m + n \log n)$ where m is the number of edges in the edge set.
- Thus, computing C, M takes $\mathcal{O}(b^2 n^4)$ time.
- Line 12 is a call to the function **FindC** which takes $\mathcal{O}(b^2 n^4)$ time.
- Lines 13-15 take $\mathcal{O}(b)$ time.
- Lines 16-23 take $\mathcal{O}(b^3)$ time because of 3 nested loops each of which iterate over $\mathcal{O}(b)$ values.
- Thus, **ComputeDP** and **FindC** together take $\mathcal{O}(b^2 n^4 + b^3)$ time.
- Note that $Parent[i][j] < i$ because it is either -1 or in line 20, l is iterated from 0 to $i - 1$. Thus, in each call of **GetRecursivePath**, $|P'| = i - l$ which is non-empty. Thus, in the recursion tree, i strictly decreases (because value of i in next iteration is l) and terminates when $Parent[i][j] = -1$. Thus, $\mathcal{O}(b)$ recursive calls are there. We can update path sequence in a global array, copying each element in this array would take $\mathcal{O}(n)$ time (because path can be of length atmos $n - 1$) and since there are $\mathcal{O}(b)$ elements, it takes $\mathcal{O}(nb)$ time. Thus, total time taken is for the function **GetRecursivePath** with $i = b, j = b$ isz $\mathcal{O}(nb^2)$
- All of the above mentioned steps take polynomial time. Complexity of the algorithm is $\mathcal{O}(b^2 n^4 + b^3)$.