CS345 Assignment 4

Abhisek Panda 150026

Abhibhav Garg 150010

October 2017

1 Question 1 - Dynamic Programming

The following operations will be used as subroutines for the main algorithm. Each of the subroutines runs in poly-time and is called polynomially many times at most.

- Finding a shortest path between two given vertices - Any known algorithm can be used, as discussed in class. Since edge weights are positive, we use a simple Dijkstras algorithm.
- Given a set of k graphs, G_i, with the same vertex set and possibly different edge sets, we can find the optimal edge weight common path(optimal in the sense that it minimizes edge weight sum over all graphs) in polynomial time as follows: Construct graph G' with the same vertices as the given graphs. For every pair of vertices i, j, add edge (i, j) in G' if and only if (i, j) is an edge in all G_i. In other words, we take the intersection of the set of edges and add these to G'. Then, find the shortest path in this G' and return this as the optimal path such that it is the same for all the given graphs. If no path exists, return infinity. This takes polynomial time since the intersection operation can be done in atmost $O(kn^2)$ time, and shortest path is poly time. This is the optimal path satisfying the given conditions because if any other path had a shorter cost that satisfied the same condition, it would exist in the intersection graph, and the shortest path algorithm would have found that instead. Therefore every other path either has same or greater length. Call this method **commonPath**.

1.1 Main Algorithm

In the spirit of DP, we build the solution from the ground up. In other words, we first assume we have the optimal solution for the first k timesteps, and use these solutions to get the optimal solution for the $k + 1$ time step.

Optimal solution for the first time step, $k = 1$: It is trivial to see that given just the first graph, the optimal solution will just be to assign to the graph its shortest path. The cost would be then just the weight of its shortest path.

Going from k to $k + 1$: Consider the optimal solution for the first $k + 1$ graphs. Let the path assigned to the $k + 1^{th}$ graph in this solution be $opt(k + 1)$. The following cases arise: either only the $k + 1$ graph has $opt(k + 1)$, and the kth graph has a different path. Or, the kth and k+1th graphs may have opt(k+1) and k-1th graph has a different path. Or the last three graphs have the same path, or the last four and so on. These are a set of mutually exclusive exhaustive events that can possibly occur. Also note that if the last l graphs have the same path, then the first $k + 1 - l$ graphs have an independent optimal solution. This lets us find opt(k+1) by checking over all possible events, and using knowledge of previous optimal solutions. We have the following relation for the optimal cost after including the $k + 1$st graph

$$optC(k + 1) = min(cP(G_1, \ldots G_{k+1}), optC(l) + K + cP(G_{l+1}, \ldots, G_{k+1})), 1 \leq l \leq k$$

where cP is the commonPath function defined above, and optC is the optimal cost solution. The actual path can be recovered by checking which argument is the minimum, and setting the path of all the graphs considered together in that argument as the best common path.

1.2 Proof Of Correctness

The proof of correctness follows a simple induction argument. The base case is trivially true. The induction step holds by construction - We minimize over a mutually disjoint exhaustive set of possibilities, and each subroutine gives independently the correct solution. This completes the proof.

2 Question 2: Analysis of Ford Fulkerson Algorithm

Given a graph G(V, E) with unit capacity edges and designated source and target vertices, s and t such that distance between s & t is $\Omega(n^{2/3})$. To prove that ford fulkerson takes $O(mn^{2/3})$ time to compute the max flow in G.

We know that the ford-fulkerson algorithm takes $O(mF)$ time to compute max flow in a graph, if F is the maximum flow value and each edge carries integral flow. The integrality is conserved in this problem since all edges have integral(=1) capacity. So, we give a bound on the value of F below:

- Perform a BFS traversal starting from s in the graph. This distinguishes all the points on the basis of their distance from s. Let all vertex at a distance of j from s belong to the set D_j. Note that the only possible edges are between the vertices of D_j or between D_j and D_{j+1},i.e, consecutive levels. This is because BFS computes the shortest paths(unit edges) and jumping levels will lead to violation of shortest path property.
- Let the length of the path between s & t be $kn^{2/3}$, where k is a constant > 1. This implies that number of levels before $t = kn^{2/3}$.
- Consider sets of sizes atleast $\frac{2}{k}n^{1/3}$. Let there be l such sets. The upper bound on l can be found by comparing with totla number of vertices in the graph:

$$l * \frac{2}{k}n^{1/3} < n \implies l < \frac{k}{2}n^{2/3}$$

  . The strict inequality follows since $D_0 = s$ which is a singleton set.
- Thus, by pigeon hole principle, there exist atleast 2 consecutive levels before t such that each set has size less than $\frac{2}{k}n^{1/3}$. The maximum number of forward edges between these 2 levels is $(\frac{2}{k}n^{1/3})^2$ in worst case, which is $O(n^{2/3})$.
- Define a cut between the above 2 levels. This cut will have a flow value of $O(n^{2/3})$, when all the forward edges have unity flow. Since the max flow value is bounded by the value of any cut, $F = O(n^{2/3})$.

Therefore, the time complexity of the ford-fulkerson algorithm turns out to be $O(mn^{2/3})$ under the constraints imposed.

3 Question 3 - Network Flow Application

3.1 Part A:

To formulate the given problem instance as a max-flow problem, the following constraints are noted:

- Each balloon can measure atmost 2 readings.
- Each balloon measures readings only from a set of conditions, S_i.
- Each condition has to be measured by k distinct balloons.

Therefore, we construct the network as follows:

- Construct a bipartite graph, with edges from B(set of balloons) to C(set of conditions). Each node i in B has edges to nodes that belong to S_i(constraint 2) in C. Each of these edges will have a flow capacity of 1 since each balloon needs to measure a condition atmost once.
- Add a source vertex s with edges to all nodes in B. These edges will have a flow capacity of 2 each, since each balloon is allowed to take atmost 2 measurements.
- Add a target vertex t with incoming edges from each node in C. These edges will have a flow capacity of k each, since we require each condition to be measured atleast k times.

Once we have constructed the network, there exists a way of measuring the conditions under the imposed constraints only if the maximum flow is $k|C| = kn$. This follows from the observation that a cut between C and t can have a maximum flow of kn only when all edges are saturated. Saturated edges implies a flow of k through each edge from C to t which is exactly the required constraint 3. Note that constraint 1 and 2 are already satisfied due to the way we constructed the network.

3.1.1 Time Complexity:

The construction of the graph will require $O(|E|)$ time where $|E|$ is the total number of edges in the network.

$$|E| = m + n + \sum_{i=1}^{i=m} |S_i|$$

After construction, we can determine the max flow in network in $O(|E|F)$ time where F is the max flow, which in worst case is kn.

3.2 Part B:

The additional constraint is that all k readings of a particular condition should not come from balloons manufactured by the same industry(out of 3 such industries). This implies that the flow of k units received from a condition should bee distributed among the 3 industries in such a way that no one receives a flow of k units. This can be ensured by limiting the capacity of these connections at $(k-1)$. Thus, the new network is as follows:

- Construct a bipartite graph, with edges from B(set of balloons) to C(set of conditions). Each node i in B has edges to nodes that belong to S_i(constraint 2) in C. Each of these edges will have a flow capacity of 1.
- Add a source vertex s with edges to all nodes in B. These edges will have a flow capacity of 2 each.
- Introduce a dummy layer D, with one node corresponding to each node in C. Connect each node in C with its corresponding node in D with an edge of flow capacity k.
- Introduce a layer I with 3 nodes for the 3 industries. Connect each node in D with each node in I(fully connected) with edges of capacity $(k-1)$.

- Finally, connect all nodes in I with a target node with edges having very large capacity(can be ∞) since there is no constraint as to how many balloons one industry can manufacture.

As before, the problem again reduces to finding whether there exists a max flow of value kn in the network. The cut corresponding to the max flow is betweeen layers C & D. Note that all other constraints are satisfied by the manner in which we constructed the network.

3.2.1 Time Complexity:

$$|E| = m + n + \sum_{i=1}^{i=m} |S_i| + n + 3n + 3$$

which is polynomial in m & n. So, time complexity is $O(|E|F)$ where $F = kn$ in worst case.