Analysis of Algorithms - Home Work 4

CSE 548 Fall ’19

Prof. Jie Gao

Submission By:

Narayan Acharya

112734365

Contents

1 Question 1

Textbook [Kleinberg & Tardos] Chapter 7, page 415, problem #6.
Solution:

Consider the following notation: We denote the set of all switches by A and the set of all the fixtures by B. We wish to pair every item in set A with at least one item in set B such that the entire setup of switches and fixtures is ergonomic. A setup is ergonomic if we can pair every fixture to a switch such that the line joining each pair is not interrupted by any wall. The problem is one where we wish to find perfect matching between in the bipartite graphy constructed using the two sets A and B.

Let us construct a bipartite graph $G = (V, E)$ where $V = A \cup B$ and E is the set of all the edges from A to B if a switch can be connected to a fixture without being interrupted by a wall.

Now we can find perfect matching, if one exists, using the Max Flow Algorithm. For this consider we have source node s that initiates a flow of n over n edges of capacity 1, one to each of the nodes in set A. We also have a sink node t that accepts flow of n over n edges of capacity 1 from each of the nodes in set B.

The correctness and time complexity can be inferred from *7.38 in Kleinberg and Tardos (Page 370)*. We can say that the time complexity is $O(mn)$ where m is the number of edges and n is the number of nodes in the bipartite graph. Thus we have an algorithm with time complexity polynomial in m and n. In our case m is bound by n^2 as there is a possibility of every switch having a matching with every fixture. Thus the total running time of our solution will be bound by $O(n^3)$.

2 Question 2

Textbook [Kleinberg & Tardos] Chapter 7, page 415, problem #11.
Solution:

The claim from the friends is false and can be proved so using a counter-example.
In order to construct a counter-example build a bipartite graph as follow - Create a set of b nodes, say A, and another set of b nodes, say B. Number nodes in both A and B from 1 to b. Add edges such that the i^{th} node in A connects to the i^{th} node in B and the i^th node in B connects to the $(i+1)^{th}$ node in A. All edges are of capacity 1.

Next, add a source node, s, and a sink node, t. Add b edges of capacity 1 from s, one to each of the nodes in A and b of capacity 1, one from each of nodes in B to t.

Now, if we select the augmenting path in the first path to be $s \to A_1 \to B_1 \to A_2 \ldots A_b \to B_b \to t$ we have a flow value of 1 from s to t. Next, as per the Forward-Only Algorithm suggested, we do not consider any of the backward edges from the residual graph so constructed. But if we do so, we are left with no path from s to t in the residual graph. Thus, we end and claim that the max flow is 1. But, clearly the max flow for our bipartite graph is b. Thus, we have failed to reach our potential.

Given, b is expected to be independent of the structure of our bipartite graph and we can choose our b to be absurdly large but our flow will always max out at 1. Thus, this proves our case for the counter-example. Hence, we can say that the claim is false!

3 Question 3

Textbook [Kleinberg & Tardos] Chapter 7, page 415, problem #13.
Solution:

In order to apply known and applicable algorithms for Max Flow, we make the following changes to our graph, say G, without lifting any of the restrictions posed by the question, to create a new graph, say G'.

1. Replace every node, say v, with two nodes, say v_i and v_o. The edge between the two new nodes will have a capacity of c_v which is the initial capacity of our node v. The edge coming into v will now be connected to v_i and the edge coming out of v wi now be coming out of v_o. The capacities of these edges are unchanged.
2. All other edges can be assumed to have an edge capacity of ∞

Next, in order to ensure that we can apply the Max-Flow algorithm to G' and get the same flow as G we need to prove that the G' follows both the edge capacity constraint as well as flow conservation constraint.

1. Capacity Constraint: For each node v, we know that $c_v \geq 0$. Thus the edges that we create have positive edge capacities equal to c_v. Thus the flow, $f(v_i \rightarrow v_o)$, going through this new edge will obey the capacity constraint, $0 \leq f(v_i \rightarrow v_o) \leq c_v$.
2. Flow Conservation Constraint: The edge we added in G' is bound by c_v. Any outgoing flow from v_i cannot exceed c_v. This ensure that any incoming flow into v_o will not exceed c_v. So, overall whatever comes into v_i will be going through v_o and out of it, bound by c_v. This ensures, flow is conserved even in G'.

Thus, we can say that the maximum flow is bound by the capacity if the minimum capacity cut in both G and G'. Say we use Edmond-Karp algorithm, we have an algorithmic time complexity of $O(mn^2)$ where m is the number of edges and n is the number of nodes in our graph G. But given that we've introduced n more edges after we replaced our nodes with 2 nodes in our node-capacitated network, the running time is now $O((m+n)n^2)$, which is polynomial in m and n.

4 Question 4

Textbook [Kleinberg & Tardos] Chapter 7, page 415, problem #15.
Solution:

Construct a bipartite graph as follows - Create set of nodes, say P, one node identifying each of n people involved. Create another set of nodes, say D, one node identifying each of the days for which the persons are expected to be cooking. We add an edge from a node in P to a node in D based on the person's availability to cook on that day.

Next we consider the partial solution that we have from Alanis. The assignment of people to days from Alanis can be considered to be an $n-1$ matching because for a particular day d_l we have no assignments because 2 people, p_i and p_j were assigned the same day, d_k.

In order to correct the schedule we need to build up from the partial assignments that we already have. This can be done by the following process.

1. Remove edge from either p_i to d_k or p_j to d_k. Thus, we no longer have n edges but have $n-1$ edges depicting the $n-1$ correct matchings that we have so far.
2. Add a source node, s, that initiates a flow of value n over n edges of edge capacity 1, one to each of the nodes in P. Add a sink node, t, that accepts a flow of n over n edges of capacity of 1 each, from the n nodes in D.
3. Construct residual graph from the current state of the graph that has s, P, D and t.
4. Look for an augmenting path. If we manage to find an augmenting path then we can push our matching from $n-1$ to n. This will be our perfect matching.

The running time of this process is only bound by the time it will take to find the augmenting path. We can use any of the techniques discussed in class to get this. The running time is essentially bound by $O(m)$. In our case, m can take a maximum value of n^2 for there can be a maximum of n^2 edges between P and D. Thus the running time of the algorithm is as required, $O(n^2)$.

5 Question 5

Textbook [Kleinberg & Tardos] Chapter 7, page 415, problem #16.
Solution:

Construct a bipartite graph as follows - Create a set of nodes, say P, one node identifying each of the n people who are viewers of the ads by advertisers. Create another set of nodes, say A, one identifying each of the the m advertisers who are showing these ads. We draw an edge from the a person to an advertiser, based on person's registration information if they belong to a certain demographic for which the advertiser has a targeted ad.

Next, we create a source node, s, from which we have initiate a flow of value n, over n edges each with a capacity of 1. We also create a sink node, t, which accepts m incoming edges. The only constraint here is that for each of these m edges, we need to enforce a minimum capacity or lower bound constraint of value of r_i for the i^{th} advertiser. This condition will help us ensure that contract with each of the advertisers is upheld.

This means that our problem is now similar to a *Circulation with Demands problem [See Kleinberg & Tardos, page 382]*. Here, our demand is modeled by the value r_i for the i^{th} advertiser. The demand on the source node, s, if $-\Sigma r_i$ and that on the sink node, t, if $-\Sigma r_i$ will ensure that feasible circulation exists. If a feasible circulation exists it implies that all our constraints are taken care of, which includes the minimum number of users targeted for a particular minute for each of the m advertisers.

6 Question 6

Textbook [Kleinberg & Tardos] Chapter 7, page 415, problem #24.
Solution:

Consider the following steps in order to determine if a graph, say G, has a unique minimum cut or not.

1. Apply Max-Flow on G to then infer the minimum s-t cut by applying any graph traversal algorithm on the residual graph G_f for max flow graph. Say the max flow obtained is f.
2. Split the vertices into two sets, S and T such that source, s, and all nodes reachable from s go into S while all other nodes including t go to set T.
3. Increase the edge capacities of all the edges on the s-t cut by one to create a new graph G'.
4. Calculate max flow for G' and call it f'.
5. If $f' = f$, the s-t cut obtained in the first step is not the unique minimum s-t cut as there exists another set of edges that can achieve the same flow value.
6. If $f' > f$, every time we add 1 to the edge capacities of the edges on the s-t cut, the s-t cut obtained in the first step is the unique minimum s-t cut.

Running time for the above algorithm is a linear summation of calculating the max-flow and then use graph traversal, say BFS and do this number of times. Max-flow can be calculated in polynomial time in m and n where m is the number of edges while n is the number of nodes in the graph G. BFS, too, can be done in polynomial time in m. The number of time we run this is bound by $m+1$ in order to isolate if there is a unique minimum cut possible or not. Thus, our algorithm is run in polynomial time.

References

[1] Amanpreet Singh,
`Discussed approaches with Amanpreet Singh (SBU ID - 112044129)`

[2] Atharva Urdhwareshe,
`Discussed approaches with Atharva Urdhwareshe (SBU ID - 112671765)`