Solutions to Problem Set 5

Rashmi Dwaraka

November 13, 2016

Collaborator: Manthan Thakar, Nakul Camasundaram, Alankrith Joshi

Contents

1 Bottleneck bandwidth

Problem statement:
One can model a communication network by a weighted directed graph G, in which the vertices of G represent the nodes of the network, the edges of G represent the communication links of the network and the weight of an edge is the bandwidth of the link. We define the bottleneck bandwidth of a path p as the bandwidth of the minimum-bandwidth link in p.

Design and analyze an efficient algorithm to determine a path with the largest bottleneck bandwidth from a given node s to a given node t. If no path from s to t exists, then your algorithm must indicate so. Prove the correctness of your algorithm. Analyze its worst-case running time. You may assume that the bandwidth of every link is positive. (Hint: Modify Dijkstras algorithm.)

Solution:
Given G = (V,E), where V is a set of nodes of the network and E is a set of edges representing the communication links of the network. The weight of the link indicates the bandwidth of the link. The algorithm takes in the Graph (G), source node (s) and destination node (t).
We can modify the Dijkstra's RELAX algorithm (mentioned in pg 649 of CLRS) as below. The attribute d of t vertex will represent the minimum bandwidth link. If there is no path from source to destination, then t.d will not be updated.

Algorithm: INITIALIZE-SINGLE-SOURCE(G, s, t)

```
1: procedure: INITIALIZE-SINGLE-SOURCE(G, s, t)
2:     for each vertex v in V
3:         v.d = -∞
4:         v.p = NIL
5:     s.d = ∞
```

Algorithm: BOTTLENECK-BANDWIDTH(G, s, t)

```
1: procedure: BOTTLENECK-BANDWIDTH(G, s, t)
2:     INITIALIZE-SINGLE-SOURCE(G, s, t)
3:     S = ∅
4:     Q = G.V
5:     while Q ≠ ∅
6:         u = EXTRACT-MAX(Q)
7:         S ∪ {u}
8:         for each vertex v ∈ G.Adj[u]
9:             RELAX(u,v,w)
10:    if t.d == -∞
11:        return NO-PATH
12:    else return t.d
```

Algorithm: RELAX(u, v, w)

```
1: procedure: RELAX(u, v, w)
2:         if v.d < min(u.d, w(u, v))
3:             v.d = min(u.d, w(u, v))
4:             v.p = u
```

Correctness of the Algorithm
We can prove the correctness of the algorithm by induction method on number of visited nodes.
Invariant hypothesis: For each visited node u, δu is the bottleneck bandwidth of path from source to u; and for each unvisited v, δv is the bottleneck bandwidth via visited nodes only from source to v (if such a path exists, otherwise -infinity).
Base case: When visited node count is one, that is the source node, the hypothesis is trivial.
Inductive hypothesis: Let us assume the hypothesis is true for n-1 visited nodes.
Inductive step: Now, let us choose an edge ($\{u, v\}$) where v has the maximum $\delta(v)$ of any univisited node and the edge ($\{u, v\}$) is such that $\delta(v) = \min(\delta(u), w(\{u, v\}))$. $\delta(v)$ must be the bottleneck bandwidth from source to v because if there were a path with better bottleneck bandwidth, and if w was the first unvisited node on that path then by hypothesis $\delta(w) < \delta(v)$ creating a contradiction. Similarly if there was a path with better bottleneck bandwidth to v without using unvisted nodes then $\delta(v)$ would have been less than $\min(\delta(u), w(\{u, v\}))$
After processing v, it still holds true that for each unvisited node w, $\delta(w)$ is the bottleneck bandwidth from source to w using visited nodes only, since if there were a different path which doesn't visit v, that would be visited previously and if there is a better bottleneck bandwidth in the path using v, we will update it when processing v.

Analysis of worst-case running time
The worst case run-time of the algorithm is $O(|V|log|V| + |E|)$ by using Fibonacci heap data structure as the modification does not include any extra run-time to Dijkstra's Algorithm

2 Shortest paths on vertex-weighted graphs

Problem statement:
Suppose you are given a connected undirected graph with weights on vertices (rather than on edges) and you are asked to compute the single-source shortest paths from a given source vertex. Here, the length of a path is defined as the sum of the weights on the vertices comprising the path. Give an algorithm for solving this problem by reducing it to the standard single-source shortest paths problem on directed graphs with weights on edges.

Solution:
We can reduce the given problem to standard single-source shortest path problem as below:
Step1: Consider a dummy vertex d whose weight is 0. Add d to the Graph G with an edge connecting to the source s.
Step2: Consider the length of an edge $\{u, v\}$ = weight(v).
Step3: Run the standard Dijkstra's algorithm with d as the source.

Run-time Analysis:
1. Building the graph requires $(|V| + |E|)$
2. Dijkstra's algorithm requires $O(|V|log|V| + |E|)$
The total running time of the algorithm of $O(|V|log|V| + |E|)$

3 Updating the maximum flow in a network

Problem statement:
You are given a directed network G with n nodes and m edges, a source s, a sink t and a maximum flow f from s to t. Assume that the capacity of every edge is a positive integer. Describe an O(m + n) time algorithm for updating the flow f in each of the following two cases. Prove the correctness of your algorithm in each case.

(a) The capacity of an edge e increases by 1.

(b) The capacity of an edge e decreases by 1.

Solution:
Given $G = (V, E)$ where V represents n nodes and E represents m edges, let f be the maximum flow from source s, sink t.

(a) **Step1:** Build a residual graph G_f on the original flow. Increase the capacity of edge e by 1.
Step2: Do BFS, and search for an augmenting path in G_f. Since augmenting path through a graph involves only positive capacity from source to sink, if we cannot find one, then the flow cannot be increased. Since we increase the capacity for only one edge, at-most there can be only one augmenting path.

Run-time Analysis:
Building a residual graph requires $O(m + n)$ and BFS requires $O(m + n)$. The total running time of the algorithm is $O(m + n)$.

(b) **Step1:** Build a residual graph G_f on the original flow. If the edge e still has residual capacity, then we can decrease the capacity of edge e by 1 without affecting the maximum flow. Do BFS, and search for an augmenting path in G_f.
Step2: If the edge e does not include residual capacity, then we add negative capacity on the edge. Do BFS, and search for an augmenting path in G_f in reverse going from t to s which includes e.

Run-time Analysis:
Building a residual graph requires $O(m + n)$ and BFS requires $O(m + n)$. The total running time of the algorithm is $O(m + n)$.

4 Selecting online advertisements

Problem statement:
Major online portals like Google and Yahoo have considerable information about the individual users based on their past interactions. This allows them to post targeted advertisements to the users. Suppose a set U of n users, labeled 1 through n, visit the portal on a particular day. The

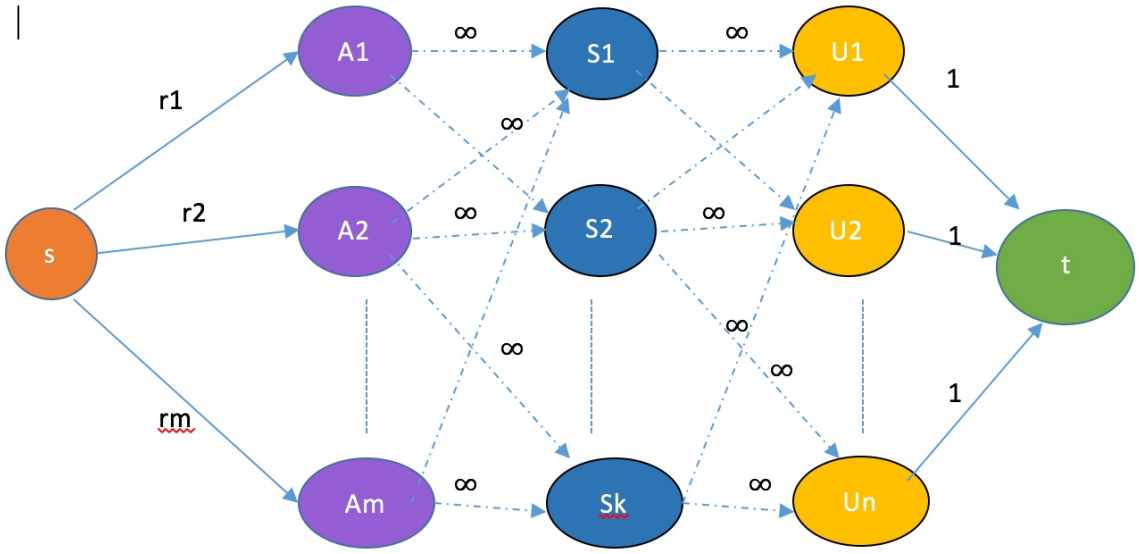
portal has a set A of m ads, labeled 1 through m, to choose from. The analysis of the users has revealed k different groups (from a marketing standpoint), the i_{th} group consisting of subset Si of users from U. A user may be part of several groups; i.e., a user may be an element of several different S_i's. The j_{th} ad is targeted to a subset $G_j \subseteq \{1, ..., k\}$ of the groups.

The portal needs to decide whether there exists a way of assigning advertisements to users such that the following conditions hold: (a) each user is shown exactly one ad; (b) ad j is shown to user i only if i is in a group k in G_j ; (c) the number of times the ad j is shown is at least r_j , where r_j is a given integer.

Give a polynomial-time algorithm that takes the above input U, A, the sets S_i's, the groups G_j's,the r_j's and determines whether the portal can assign an ad to each user so that the above three conditions are satisfied, and if so, then returns such an assignment. State the running time of your algorithm.

Solution:

Building the graph:
1. Create a node for each user $u_1, u_2....u_n$
2. Create a node for each subset S_i for k groups
3. Create a node for each element of set A of m Ads
4. Connect the user set of nodes (U) to t(sink node) with capacity 1 indicating the constraint of showing exactly 1 Ad per user
5. Connect the group subset set of nodes to user set with infinite capacity indicating the unlimited mapping between the group and user
6. Connect the Ad set (A) with group set (S) with infinite capacity indicating unlimited mapping between the group and Ad set
7. The source is connected to all the Ad set of nodes with respective r_js indicating the constraint that the number of times the ad j shown is at-least r_j times.

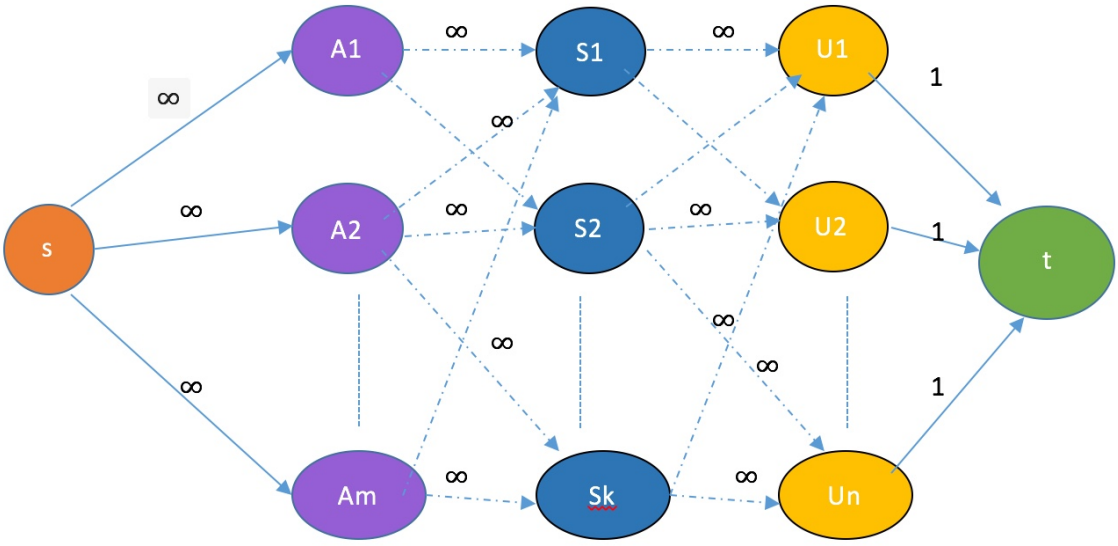


Once the graph is built, we find the max-flow of the graph to find the assignment of ads to the users.
If the max-flow "f" of the graph is less than the summation of all r_js, then the assignment fails , as (c) constraint fails.
If the max-flow "f" is equal to the summation of all r_js and it is less than n(number of users) then save the flow network as intermediate assignment solution and go to "Rebuilding the graph". Otherwise we have the final assignment of ads.

Re-building the graph:

If the max-flow in the above iteration is less than n:
This means that there are still some users , who have not been assigned any ads. Thus, we will rebuild the graph as below in order to satisfy the exactly one ad per user constraint.
In the above iteration, we satisfy the (b) and (c) constraint. For cases such as the number of users are greater than summation of the r_j for all ads, the above network will not suffice. In order to fulfill this constraint, we need to set the capacity as infinity for the edges between source and ads. Also restrict the set of users to subset of users who are not assigned any ads during the first iteration.

1. The capacity of source connected to all the Ad set of nodes is updated with infinity.



We can find max-flow for the above graph and satisfy the (a) constraint with the above graph. Combine the network flow obtained with the intermediate network flow found above. We will have a network which satisfies all the constraints. The infinite capacity between source and ad ensures all the users receive the ad. The capacity of 1 set for the edges between users and sink node ensures exactly one assignment.

Run-time Analysis:

1. Building the graph requires 2 * $O(|V| + |E|)$ where V is $(m + n + k)$
2. Combining the 2 result set $O(|V| + |E|)$
3. Finding the max-flow using Edmonds-Karp algorithm requires 2* $O(|V||E|^2)$

The total running time of the algorithm of $O(|V||E|^2)$