CS 577: HW 6

Daniel Ko

Summer 2020

§1 Give an efficient algorithm to decide if this is possible, and if so, to actually choose an ad to show each user.

§1.1 Construct a network flow model for this problem. Clearly state the meaning of each component (node, edge, capacity) of the flow network that you construct, present your algorithm in pseudocode, and give the computing complexity analysis of your algorithm.

We construct a graph $G = (V, E)$ to model the network flow.

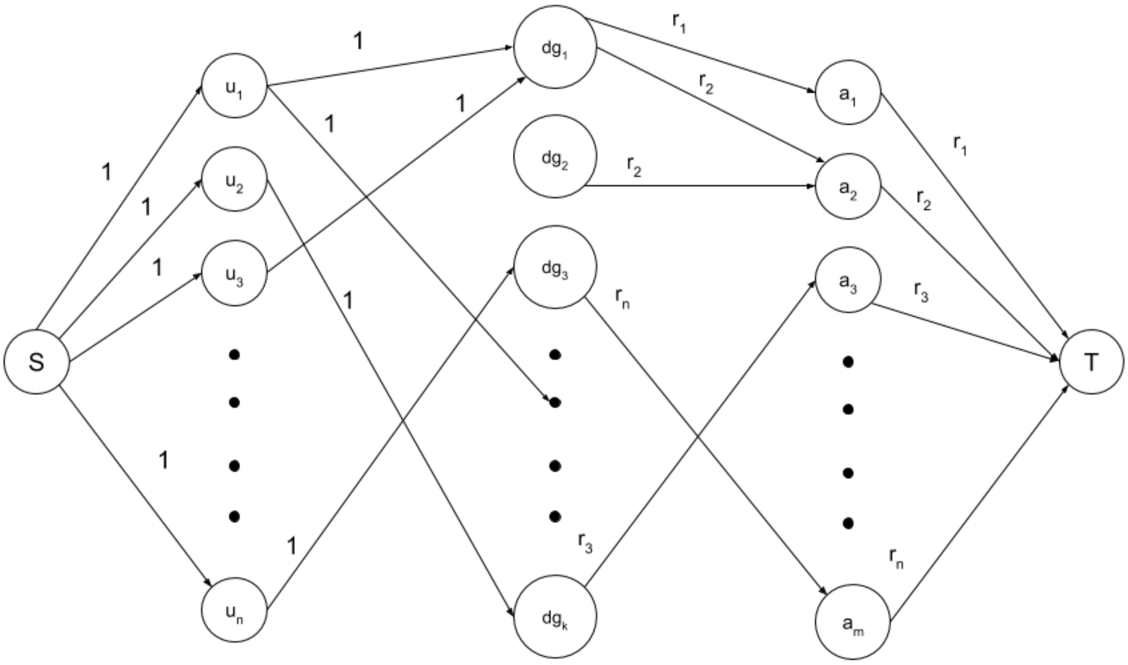
§1.1.1 Components for graph

a. **Nodes**

- Start with a source node s.
- Add a column of n nodes, denoted as u, to represent the number of users.
- Add a column of k nodes, denoted as dg, to represent the number of demographic groups.
- Add a column of m nodes, denoted as a, to represent the number of advertisers.
- End with a sink node t.

b. **Edges and capacities**

- Add an edge from source node s to each user node. Let the capacity of these edges be 1 to represent that we only want 1 ad for a user.
- Add edges from each user node to the corresponding demographic groups it belongs to. Let the capacity of these edges be 1 to represent that only 1 ad for a particular demographic groups can be shown to a user.
- Add edges from each demographic groups to each advertiser that wants to show its ads to. Let the capacity of these edges be r_i, as we only care about the lower bound of the number of users to show ads.
- Add edges from each advertiser to t. Let the capacity of these edges be r_i, as we only care about the lower bound of the number of users to show ads.



§1.1.2 Algorithm and complexity

Algorithm 1: Satisfy advertisers

```
// DG is a set containing demographic groups, where each group is
   denoted as dg_i.
// X is a set containing the set of demographic groups that each
   advertiser wants to target, denoted as X_i.
// U is a set containing the set of user's demographic groups, denoted as
   U_i
// r is a set containing the minimum number of ads that an advertisers
   wants to show to a user per minute, denoted as r_i
// m is the number of advertisers.
// n is the number of users.
function ConstructGraph(DG, X, U, r, m, n)
    Let V be a set of vertices, E be a set of edges
    Add s and t to V
    // Add users
    for u_j in U:
        Add u_j to V
        Add edge (s, u_j) to E of capacity 1
    // Add demographic group
    for dg_i in DG:
        Add dg_i to V
    for u_j in U:
        for dg in U_j:
            Add edge (u_j, dg) to E of capacity 1
    // Add advertisers
    for i from 1 to m:
        Add a_i to V
        for dg in X_i:
            Add edge (dg, a_i) to E of capacity r_i
        Add edge (a_i, t) to E from of capacity r_i
    return G = (V, E)
// The main function
function isSatisfied(DG, X, U, r, m, n)
    G ← ConstructGraph(DG, X, U, r, m, n)
    Run the Ford Fulkerson algorithm on the network flow graph G
    // Check if advertisers can be satisfied, assign user an ad
    if G has a max flow of ∑_{i=1}^{m} r_i:
        for u_i in G where f^in(u_i) = 1:
            // user u_i, from demographic dg_k will be served an ad from a_j
            print a path from u_i → dg_k → a_j that has flow 1
            decrease f^in(a_j) by one
        return True
    return False
```

We claim the time complexity to be $O(|DG|(n+m)+(n+\sum_{i=1}^{n}|U_i|+\sum_{j=1}^{k}|DG_j|+m)\sum_{i=1}^{m}r_i+km|U|)$ for `isSatisfied()`.

Proof. `ConstructGraph()` takes $O(n+|DG|+n|DG|+m|DG|)=O(|DG|(n+m))$ time. The first for loop takes n time and the second for loop takes $|DG|$ time. The third for loop takes $n|DG|$ time because there are n users and the the max amount of elements in U_j is $|DG|$ Similarly, the fourth for loop takes $m|DG|$ time. The rest of the computations take constant time.

We know in the general case the Ford Fulkerson algorithm takes, $O(Ef)$ where E is the number of edges and f is the max flow. The max number of edges in this graph is $n+\sum_{i=1}^{n}|U_i|+\sum_{j=1}^{k}|DG_j|+m$. The max flow is $\sum_{i=1}^{m}r_i$. Hence this leads to a time complexity of $O((n+\sum_{i=1}^{n}|U_i|+\sum_{j=1}^{k}|DG_j|+m)\sum_{i=1}^{m}r_i)$

Checking the if statement takes $2m$ time because computing that max flow takes m time and checking if G has such max flow takes checking all the incoming edges into T which takes m time. The for loops runs $|U|$ times and in each loop we print a path. Search for such a path using breath first search takes km time. The rest of the computations take constant time. Thus the total for loop takes $km|U|$ time and the time complexity of the this algorithm following the Ford Fulkerson takes $O(2m+km|U|)=O(km|U|)$.

Taking all these values together, the total time complexity is

$$O\left(|DG|(n+m)+(n+\sum_{i=1}^{n}|U_i|+\sum_{j=1}^{k}|DG_j|+m)\sum_{i=1}^{m}r_i+km|U|\right)$$

.

□

§1.2 Present the analysis on the correctness of your algorithm.

§1.2.1 Show that a solution to the original problem will result in flows (i.e. satisfies the conservation and capacity conditions) in the network flow graph G.

Proof. Suppose there exists an assignment of ads to users such that all advertisers are satisfied. We will show that we can obtain an $s-t$ flow from such assignment.

Let there be r_i number of user nodes, u_j, that belong to demographic group dg_λ for each advertiser a_i. For each u_j and its advertiser a_i, consider the flow that sends one unit along each path of $s\to u_j\to dg_\lambda\to a_i\to t$. Let $f(e)=1$ for each $s\to u_j$ and $u_j\to dg_\lambda$ edge. By definition, conservation condition holds for all u_j. The capacity condition holds for all $s\to u_j$ and $u_j\to dg_\lambda$ edges because the capacities of those edges is 1.

We know that $f^{in}(dg_\lambda)$ will be equal to total number of users such that all advertisers will be satisfied, i.e. the number of ads from dg_λ that a_1 wants plus the number of ads from dg_λ that a_2 wants, $\cdots$, plus the number of ads from dg_λ that a_m wants. Let $f(e)=\zeta$ for each $dg_\lambda\to a_i$, where ζ is the number of ads that a_i wants from demographic group dg_λ. Hence, conservation condition holds for all dg_λ. The capacity condition holds for all $dg_\lambda\to a_i$ because because the capacities of those edges is equal or less r_i because the total number of ads from dg_λ that a_i wants is equal or less than r_i.

Let $f(e)=r_i$ for each $a_i\to t$. Hence, conservation condition holds for all a_i. The capacity condition holds for all $a_i\to t$ because because the capacities of those edges is r_i.

Therefore, there exists an assignment of ads to users such that all advertisers are satisfied which results in a flow network. If all advertisers are satisfied, the total $s\to t$ flow will be $\sum_{i=1}^{m}r_i$. Otherwise, it will be less than the previous sum with flows less than r_i on edges $a_i\to t$. □

§1.2.2 Show that the solution to the network flow problem will give the solution for the original problem.

Proof. Suppose we have a solution with the $s \to t$ flow being $\sum_{i=1}^{m} r_i$. This means that $f(e) = r_i$ for at all edges $a_i \to t$ as the this is the only way to get total flow of $\sum_{i=1}^{m} r_i$. Consequently, this means we have identified r_i users for each a_i advertiser. Thus, all advertisers will be satisfied.

If the algorithm gives us a maximum $s \to t$ flow that is less than $\sum_{i=1}^{m} r_i$, it means $f(e) < r_i$ for at least one edge $a_i \to t$. Thus, some advertiser will not have the number of ads shown to users they requested. That is why in this case we can claim that there doesn't exist enough users such that all advertisers are satisfied. □

§2 Give a polynomial-time algorithm to decide whether it is possible to keep the set of phones fully connected at all times during the travel of this one cell phone.

§2.1 Construct a network flow model for this problem. Clearly state the meaning of each component (node, edge, capacity) of the flow network that you construct, present your algorithm in pseudocode, and give the computing complexity analysis of your algorithm.

We construct a bipartite graph $BP = (P \cup B, E)$ to model the network flow.

§2.1.1 Components for graph

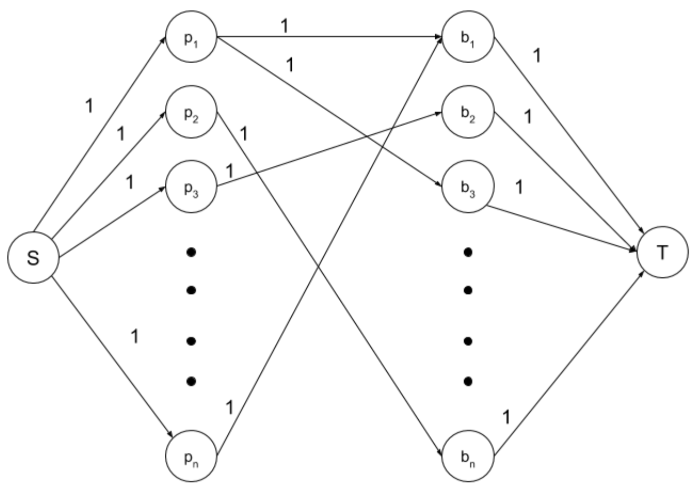
a. **Nodes**

- Add a column of n nodes, denoted as p, to represent the number of cellular phones. The set of all phones will be denoted as P.
- Add a column of k nodes, denoted as b, to represent the number of base stations. The set of all base stations will be denoted as B.

b. **Edges and capacities**

- Add edges $p_i \rightarrow b_j$ if the distance from p_i to b_j is at most Δ. Set the capacity of these edges to be 1.

The corresponding flow network G, will have a node s, with outgoing edges to all p of capacity 1, and have a node t, with incoming edges from all b of capacity 1.



§2.1.2 Naive algorithm

We can model the movement of phone p_1 by constructing a new graph it moves one unit east. Let G_0 model the state of the cellular network before any movement occurs in phone p_1. G_1 will represent the state of the cellular network where phone p_1 has moved one unit east. We update edges of p_1 such that the distance from p_1 to b_j is at most Δ given the new location. We will have a total of z graphs. If all graphs have perfect matching, there exists an assignment such that all phones will be connected during the movement of p_1.

According Theorem 7.40, we know that we can compute if a perfect match exists in $O(mn)$ time, where m is the number of edges and n is the number of nodes. We know that $m \leq n^2$ because n^2 is the maximum amount of edges we can have in this bipartite graph. Hence, computing if a perfect match exists for one graph takes $O(n^3)$ time. Since we have z graphs, the total time to compute this algorithm will take $O(zn^3)$ time.

§2.1.3 Faster algorithm

Suppose there exists a perfect match on graph G_ϕ. Notice that we can reuse the matchings we have previously computed for G_ϕ to compute matchings for $G_{\phi+1}$. If we update the edges of p_1 to state $\phi + 1$ in graph G_ϕ, the amount of matches either stays at n or decreases to $n - 1$. If the amount of matches stays at n, a perfect match exists at state $\phi + 1$ and we are done. If the amount of matches are $n - 1$, we compute a augmenting path to see if we can increase the matches to n. Computing a single augmenting path takes $O(m) = O(n^2)$.

Hence our total time complexity will be computing matches for G_0, which takes $O(n^3)$, and computing augmenting paths for the following z state changes, which takes $O(zn^2)$. This leads to a total time complexity of $O(n^3 + zn^2) = O(n^3)$, as desired.

Algorithm 2: Full connectivity

```
// P is a set containing the location of each phone, denoted as P[i]
// B is a set containing the location of each base station, denoted as P[i]
// Δ is the range parameter
function ConstructGraph(P, B, Δ)
    Let V be a set of vertices, E be a set of edges
    Add s and t to V
    // Add phones
    for i from 1 to |P|:
        Add u_i to V
        Add edge (s, u_i) to E with capacity 1
    // Add base stations
    for j from 1 to |B|:
        Add b_j to V
        Add edge (b_j, t) to E with capacity 1
    // Add edges
    for i from 1 to |P|:
        for j from 1 to |B|:
            if distance from P[i] to B[j] is at most Δ:
                Add edge (p_i, b_j) to E with capacity 1
    return G = (V, E)
// The main function
// z is units of distance that phone p_1 travels east
function fullConnectivity(P, B, z, Δ)
    S ← a list to store sequences of assignments of phones to base stations that will be
      sufficient in order to maintain full connectivity
    G ← ConstructGraph(P, B, Δ)
    Run the Ford Fulkerson algorithm on the network flow graph G
    if G has a max flow of |P|:
        S[0] ← add the current perfect matching
    else:
        print no possible matches for the initial state
    for i from 1 to z:
        G ← update p_1's edges so that it reflects its new position (1 unit east)
        if G has a max flow of |P|:
            S[z] ← add the current perfect matching
        else:
            G ← Run one iteration of Ford Fulkerson to compute an augmenting path
            if G has a max flow of |P|:
                S[z] ← add the current perfect matching
            else:
                print no possible matches for the zth state
    return S
```

Time complexity for `fullConnectivity()` is $O(n^3)$.

Proof. `ConstructGraph()` takes $2n + n^2$ time, as there is two for loops that runs n times and one nested for loop that runs n^2 times total. This leads to a time complexity for this function to be $O(2n + n^2) = O(n^2)$

As explained previously in 2.1.3, Ford Fulkerson takes $O(n^3)$ time.

The first if else statement takes n times to check max flow, as we are checking all edges that come into t. The for loop runs for a total of z times. The if statement takes n times as previously explained. The else statement takes $n^2 + n$ as we are running one iteration of Ford Fulkerson (finding a single augmenting path) and checking max flow. Hence, the time complexity for the for loop is either $O(z(n))$ or $O(z(n^2 + n))$. We take the higher complexity, leading to $O(z(n^2 + n)) = O(zn^2 + zn) = O(zn^2)$

Combining all the complexities of this algorithm leads to $O(n^2) + O(n^3) + O(zn^2)$. Since z should not change with the number of device n, it can be treated as a constant. Therefore, the total time complexity is $O(n^3)$. □

§2.2 Present the analysis on the correctness of your algorithm.

§2.2.1 Show that a solution to the original problem will result in flows (i.e. satisfies the conservation and capacity conditions) in the network flow graph G.

Proof. Suppose there exists an assignment of edges such that all phones will be connected to a base station. We will show that we can obtain an $s - t$ flow from such assignment.

For each phone p_i and the base station b_j it's connected to, consider the flow that sends one unit along each path of $s \to p_i \to b_j \to t$. Let $f(e) = 1$ for all edges. The capacity condition holds for all edges because we defined the capacities to be 1. The conservation condition holds for p_i and b_j because the flow entering and leaving the nodes is both 1.

Therefore, there exists an assignment of edges such that all phones will be connected to a base station which results in a flow network. If all phone are connected, the total $s \to t$ flow will be n. Otherwise, it will be less than the n with empty flows on edges $b_j \to t$. □

§2.2.2 Show that the solution to the network flow problem will give the solution for the original problem.

Proof. Suppose we have a solution with the $s \to t$ flow being n. This means that $f(e) = 1$ for at all edges $p_i \to b_j$ as the this is the only way to get total flow of n. Consequently, this means we have identified a base station for all phones. Thus, all phone can be connected.

If the algorithm gives us a maximum $s \to t$ flow that is less than n, it means $f(e) < 1$ for at least one edge $p_i \to b_j$. Thus, some phones cannot connect to a base station. That is why in this case we can claim that there doesn't exist stations such that all phones can be connected. □