CSE 202: Design and Analysis of Algorithms **(Due: 11/16/19)**

Homework #3

Instructor: Ramamohan Paturi *Name:* Shihan Ran, *Netid:* A53313589

Problem 1: Graph cohesiveness

Problem Description

In sociology, one often studies a graph G in which nodes represent people and edges represent those who are friends with each other. Let's assume for purposes of this question that friendship is symmetric, so we can consider an undirected graph.

Now suppose we want to study this graph G, looking for a "close-knit" group of people. One way to formalize this notion would be as follows. For a non-empty subset S of nodes, let $e(S)$ denote the number of edges in S-that is, the number of edges that have both ends in S. We define the *cohesiveness* of S as $e(S)/|S|$. A natural thing to search for would be a set S of people achieving the maximum cohesiveness.

(Subproblem 1)

Give a polynomial-time algorithm that takes a rational number α and determines whether there exists a set S with cohesiveness greater than α.

Solution

(High-level description)

We represent G as (V, E). Then we constructed a directed graph $G' = (N, E')$. It has vertex N_v for every vertex $v \in V$, vertex N_e for every edge $e \in E$, and two distinguished vertex source s and sink t.

Assume $c(N_i, N_j)$ is the capacity of edge (N_i, N_j). We let $c(s, N_e) = 1$ for every edge $e \in E$, $c(N_v, t) = \alpha$ for every vertex $v \in V$, $c(N_e, N_i) = c(N_e, N_j) = \infty$ if $e = (i, j) \in E$ and $i, j \in V$.

To determine whether there exists a set S with cohesiveness greater than α is to determine if there is an s-t cut in graph G' of capacity at most $|E|$. It's a minimum cut problem. We use the maximum-flow algorithm (e.g. Ford-Fulkerson) to find an s-t cut of minimum capacity and compare it with $|E|$.

(Correctness)

We define an s-t cut as a partition (A, B) of N with $s \in A, t \in B$. The above-defined network G' has the following properties:

- The source node s has $|E|$ outgoing edges and their capacity is all equal to 1. So the capacity of the min cut should be less than or equal to $|E|$ (we can simply get this value by letting A=s).
- Since α is a rational number. We can transform our graph by multiplying all edge weights by the same constant factor β, where β is defined to be the smallest positive number such that $\alpha\beta$ is an integer. Then, all capacities are scaled to integers.
- If N_e is in A where $e = (i, j)$, then N_i and N_j should also be in A. This is because $c(N_i, N_j) = \infty$ and infinite capacity edge can not cross a min cut since our min cut is less than or equal to $|E|$. Similarly, if node N_v is in B, then all N_e where $e = (v, j)$ for $v, j \in V$ must also be in B.

We use A_e to denote the vertices N_e in A and A_v to denote the vertices N_v in A. Then $|A_e|$ should be exactly the number of edges in the original graph G that have both endpoints in A_v, and according to the problem description, there are $e(A_v)$ such edges. The edges that cross our cut are

- All edges (N_v, t) for $N_v \in A_v$, each with the capacity α. There are $|A_v|$ edges.
- All edges s, N_e for which $e = (i, j)$ and $i, j \notin A_v$, each has a capacity 1. There are $|E| - e(A_v)$ edges.

Hence, the total capacity of our cut is $c(A, B) = \alpha|A_v| + |E| - e(A_v)$. We can arrange this to get

$$|E| - c(A, B) = e(A_v) - |A_v|\alpha$$

According to the problem description and based on the above equation, we can see that $e(A_v)/|A_v| > \alpha$ iff $|E| - c(A, B) > 0$. That is $c(A, B) < |E|$.

So we've prove that we have a min cut with $c(A, B) \leq |E|$ then we can find a group of vertices in our original graph with cohesiveness greater than α.

(Time complexity)

- There are $O(|V|^2)$ nodes.
- There are $O(|V|^2)$ edges.
- C can be arbitrarily large.
- Because we don't have a bound on C and the problem specifically asks for a polynomial-time algorithm, we don't want something that depends on C. We are thus limited to Edmonds-Karp and Preflow-push. With our knowledge of this problem, we see that preflow-push scales better.
- By using the Preflow-push algorithm to determine the minimum cut for a given α, our total complexity is $O((|V|^2)^2|V|^2) = O(|V|^6)$.

(Subproblem 2)

Give a polynomial-time algorithm to find a set S of nodes with maximum cohesiveness.

(High-level description)

There are $|V|$ choices for the size of A_v and $C_2^{|A_v|}$ choices for $e(A_v)$. Thus, it's a total of $O(|V|^3)$ possible values for α. We can find the set of maximum cohesiveness by binary search on α by checking if there exists a subset with cohesiveness greater than a given value $|E|$.

(Time complexity)

Using binary search on α takes $O(\log |V|^3) = O(\log |V|)$ time complexity. Checking if there exists a subset with cohesiveness greater than a given value $|E|$ takes $O(|V|^6)$.

Thus, the total complexity should be $O(\log |V| * |V|^6)$.

Problem 2: Remote Sensors

Problem Description

Devise as efficient as possible algorithm for the following problem. You have n remote sensors s_i and $m < n$ base stations B_j. For $1 \leq j \leq m$, base station B_j is located at (x_j, y_j) in the two-dimensional plane. You are given that no two base-stations are less than 1 km apart (in standard Euclidean distance, $\sqrt{\left((x_j - x_k)^2 + (y_j - y_k)^2\right)}$). All base stations have the same integer bandwidth capacity C.

For $1 \leq i \leq n$, sensor s_i is located at (x_i, y_i) in the two-dimensional plane and has an integer bandwidth requirement of r_i, which can be met by assigning bandwidth on multiple base stations. Let $b_{i,j}$ be the amount of bandwidth assigned to sensor s_i on base station B_j. The assignment must meet the following constraints:

- No sensor may be assigned any bandwidth on a base station more than 2 km distance from it, i.e., if the distance from s_i to B_j is greater than 2, $b_{i,j} = 0$.
- The sum of all the bandwidth assigned to any remote sensor s_i must be at least r_i: for each $1 \leq i \leq n$, $\sum_j b_{i,j} \geq r_i$.
- The sum of all bandwidth assigned on base station B_j must be at most C: for each $1 \leq j \leq m$, $\sum_i b_{i,j} \leq C$.

Your algorithm should find a solution meeting the above constraints if possible, and otherwise output a message saying "No solution exists". Prove the correctness of your algorithm and discuss its time complexity.

Solution

(High-level description)

We should start by listing key properties of the problem.

- n remote sensors s_i, $\forall i \in \{1, 2, \ldots, n\}$, located at (x_i, y_i), with an integer bandwidth requirement of r_i met by assigning bandwidth on multiple base stations
- m base stations B_j, $\forall j \in \{1, 2, \ldots, m\}$, located at (x_j, y_j), with the same integer bandwidth capacity C
- No two base stations B_j and B_k are less than 1 km apart
- No sensors may be assigned any bandwidth on a base station more than 2 km distance from it
- $b_{i,j}$ be the amount of bandwidth assigned to sensor s_i on base station B_j
- $\forall i \in \{1, 2, \ldots, n\}$, $\sum_j b_{i,j} \geq r_i$
- $\forall j \in \{1, 2, \ldots, m\}$, $\sum_i b_{i,j} \leq C$

We want to find the assignment of bandwidth, which suggests a matching problem, and thus a max-flow problem. We constructed a directed graph $G = (V, E)$.

- Nodes
  - s, t
  - m nodes representing the base stations: B_j, $\forall j \in \{1, 2, \ldots, m\}$
  - n nodes representing the remote sensors: s_i, $\forall i \in \{1, 2, \ldots, n\}$
- Edges
  - $s \to B_j$, $\forall j \in \{1, 2, \ldots, m\}$, with capacity C, which ensures that each base station can have at most C incoming flow
  - $B_j \to s_i$, $\forall i, j$ where sensors s_i and base stations B_j are less than 2 km apart, with capacity C. This ensures that any given base station can only contribute to sensors that it can do.

- $s_i \to t$, $\forall i \in \{1, 2, \ldots, n\}$, with capacity r_i

The solution is when the maximum flow in this network G saturates all incoming edges of t. Otherwise, the assignment does not exist. As matchings, output the $B_j \to s_i$ flow.

(Correctness)

- Base stations have incoming flow at most C, thus ensuring that the sum of all bandwidth assigned on base station be at most C
- Remote sensors have outgoing flow at most r_i, and it has been saturated, so the sum of all the bandwidth assigned to any remote sensor s_i is r_i
- $B_j \to s_i$ edges exist only if the sensors s_i and base stations B_j are less than 2 km apart, so incompatible matches won't arise
- Each matched base stations contributes a bandwidth of $b_{i,j}$, which ensures the following
  - If the $s_i \to t$ is saturated, then s_i satisfies the bandwidth requirement of r_i
  - If s_i satisfies the bandwidth requirement of r_i, then that means the $s_i \to t$ is saturated
  - The saturated $s_i \to t$ edges correctly correspond to the matchings

(Time complexity)

- There are $O(n + m) < O(2n) = O(n)$ nodes.
- There are $O(n)$ edges since no two bases-stations are less than 1 km apart and no sensor may be assigned by any bandwidth on a base station more than 2 km distance from it.
- With our knowledge of this problem, we see that preflow-push scales better. By using the Preflow-push algorithm to determine the maximum flow, our complexity is $O((n)^2 n) = O(n^3)$.

Problem 3: Scheduling in a medical consulting firm

Problem Description

You've periodically helped the medical consulting firm Doctors Without Weekends on various hospital scheduling issues, and they've just come to you with a new problem. For each of the next n days, the hospital has determined the number of doctors they want on hand; thus, on day i, they have a requirement that exactly p_i doctors be present.

There are k doctors, and each is asked to provide a list of days on which he or she is willing to work. Thus doctor j provides a set L_j of days on which he or she is willing to work.

The system produced by the consulting firm should take these lists and try to return to each doctor j a list L'_j, with the following properties.

(A) L'_j, is a subset of L_j, so that doctor j only works on days he or she finds acceptable.

(B) If we consider the whole set of lists $L'_1, \ldots, L'_k$, it causes exactly p_i doctors to be present on day i, for $i = 1, 2, \ldots, n$.

(Subproblem 1)

Describe a polynomial-time algorithm that implements this system. Specifically, give a polynomial-time algorithm that takes the numbers $p_1, p_2, \ldots, p_n$, and the lists $L_1, \ldots, L_k$, and does one of the following two things.

- Return lists $L'_1, \ldots, L'_k$ satisfying properties (A) and (B); or
- Report (correctly) that there is no set of lists $L'_1, \ldots, L'_k$ that satisfies both properties (A) and (B).

(Solution 1)

(High-level description)

We should start by listing key properties of the problem.

- n days
- k doctors
- On day i they have a requirement that exactly p_i doctors be present, $\forall i \in \{1, 2, \ldots, n\}$
- Doctor j provides a set L_j of days on which he or she is willing to work and he or she only works on days he or she finds acceptable

Apparently, if we let q_i be the number of doctors willing to work on day i, and for any $i \in \{1, 2, \ldots, n\}$, we have $q_i < p_i$, then there cannot be a set of lists $L'_1, \ldots, L'_k$ satisfies property (B). If $\forall i \in \{1, 2, \ldots, n\}$, we have $q_i \geq p_i$, then we can just arbitrarily pick p_i doctors out of those q_i doctors to work on day i.

(Correctness)

This algorithm is correct since it ensures that

- All doctor only works on days he or she finds acceptable (since on day i, we only consider doctors who are willing to work)
- On day i, it cause exactly p_i doctors to be present (since we pick p_i doctors out of those q_i doctors to work on day i)

(Time complexity)

For each day $i \in \{1, 2, \ldots, n\}$, we should check how many doctors are willing to work. Thus, the total time should be $O(nk)$.

(Subproblem 2)

The hospital finds that the doctors tend to submit lists that are much too restrictive, and so it often happens that the system reports (correctly, but unfortunately) that no acceptable set of lists $L'_1, \ldots, L'_k$ exists.

Thus the hospital relaxes the requirements as follows. They add a new parameter $c > 0$, and the system now should try to return to each doctor j a list L'_j with the following properties.

(A*) L'_j contains at most c days that do not appear on the list L_j.

(B) (Same as before) If we consider the whole set of lists $L'_1, \ldots, L'_k$, it causes exactly p_i doctors to be present on day i, for $i = 1, 2, \ldots, n$.

Describe a polynomial-time algorithm that implements this revised system. It should take the numbers $p_1, p_2, \ldots, p_n$, the lists $L_1, \ldots, L_k$, and the parameter $c > 0$, and do one of the following two things.

- Return lists $L'_1, \ldots, L'_k$ satisfying properties (A*) and (B); or
- Report (correctly) that there is no set of lists $L'_1, \ldots, L'_k$ that satisfies both properties (A*) and (B).

(Solution 2)

(High-level description)

We should start by listing key properties of the problem.

- n days
- k doctors
- On day i they have a requirement that exactly p_i doctors be present, $\forall i \in \{1, 2, \ldots, n\}$
- Doctor j provides a set L_j of days on which he or she is willing to work but he or she can work on at most c days that do not appear on the list L_j

This time the problem definition is a little bit different. If we let q_i be the number of doctors willing to work on day i, and for any $i \in \{1, 2, \ldots, n\}$, we have $q_i < p_i$, we can define a new variable difference $d_i = \max(0, p_i - q_i)$. We constructed a directed graph $G = (V, E)$.

- Nodes
  - s, t
  - k nodes representing the doctors: j, $\forall j \in \{1, 2, \ldots, k\}$
  - n nodes representing the days: i, $\forall i \in \{1, 2, \ldots, n\}$
- Edges
  - $s \to j$, $\forall j \in \{1, 2, \ldots, k\}$, with capacity c
  - $j \to i$, $\forall i, j$ where doctor j do not want to work on day i, with capacity 1
  - $i \to t$, $\forall i \in \{1, 2, \ldots, n\}$, with capacity d_i

This suggests a matching problem, and thus a max-flow problem. The solution is when the maximum flow in this network G saturates all incoming edges of t. Otherwise, the assignment does not exist. As matchings, output the $j \to i$ edges that have flow.

(Correctness)

This algorithm is correct since it ensures that

- For each day i, we need d_i extra doctors who do not want to work on day i (this is ensured by saturating all incoming edges of t)
- Each doctor only work on at most c days (this is ensured by setting the capacity of $s \to j$ to be c)

(Time complexity)

- There are $O(n + k)$ nodes.
- There are $O(n * k)$ edges.
- With our knowledge of this problem, we see that preflow-push scales better. By using the Preflow-push algorithm to determine the maximum flow, our complexity is $O((n + k)^2 nk)$.
- Calculating d_i needs $O(nk)$ time complexity. Thus, the total time complexity is $O(nk + (n + k)^2 nk) = O((n + k)^2 nk)$.

Problem 4: Cellular network

Problem Description

Consider the problem of selecting nodes for a cellular network. Any number of nodes can be chosen from a finite set of potential locations. We know the cost $c_i \geq 0$ of establishing site i. If sites i and j are selected as nodes, then we derive the benefit b_{ij}, which is the revenue generated by the traffic between the two nodes. Both the benefits and costs are non-negative integers. Find an efficient algorithm to determine the subset of sites as the nodes for the cellular network such that the sum of the benefits provided by the edges between the selected nodes less the selected node costs is as large as possible.

Design an efficient polynomial-time algorithm.

Provide a high-level description of your algorithm, prove its correctness, and analyze its time complexity.

Solution

(High-level description)

Let n be the total number of sites. We constructed a directed graph $G = (V, E)$.

- Nodes
  - s, t
  - n nodes representing sites: S_i, $\forall i \in \{1, 2, \ldots, n\}$
  - n^2 nodes representing benefit: B_{ij}, $\forall i \in \{1, 2, \ldots, n\}, j \in \{1, 2, \ldots, n\}$
- Edges
  - $s \to B_{ij}$, $\forall i \in \{1, 2, \ldots, n\}, j \in \{1, 2, \ldots, n\}$, with capacity b_{ij}
  - $B_{ij} \to S_i$ and $B_{ij} \to S_j$, $\forall i \in \{1, 2, \ldots, n\}, j \in \{1, 2, \ldots, n\}$, with capacity ∞. We do not want these edges to be cross the cut.
  - $S_i \to t$, $\forall i \in \{1, 2, \ldots, n\}$, with capacity c_i

We define an s-t cut as a partition (A, B) of V with $s \in A, t \in B$. The solution is: selecting all sites for which $S_i \to t$ is saturated (i.e., $S_i \in A$).

(Correctness)

- Consider the types of edges crossing the min-cut.
  - Case 1: $s \to B_{ij}$: $B_{ij} \in B$. Let $\sum_{ij,\text{where } B_{ij} \notin A} b_{ij}$ be these nodes' contribution to the value of the cut.
  - Case 2: $S_i \to t$: $S_i \in A$. Let $\sum_{i,\text{where } S_i \in A} c_i$ be these nodes' contribution to the value of the cut.
- Let C be the capacity of the cut. Thus, $C = \sum_{ij,\text{where } B_{ij} \notin A} b_{ij} + \sum_{i,\text{where } S_i \in A} c_i$. That is, $\sum_{i,\text{where } S_i \in A} c_i = C - \sum_{ij,\text{where } B_{ij} \notin A} b_{ij}$.
- B_{ij}, for any i, j, will only end up in A if both of S_i, S_j are in A (since the intermediate edges have infinite capacity). Thus, our gross benefit is $\sum_{ij,\text{where } B_{ij} \in A} b_{ij}$
- Our cost is $\sum_{i,\text{where } S_i \in A} c_i$, as we claimed that these are the sites that we choose.

- The total profit is

$$\begin{aligned} T &= \sum_{ij,\text{where } B_{ij}\in A} b_{ij} - \sum_{i,\text{where } S_i\in A} c_i \\ &= \sum_{ij,\text{where } B_{ij}\in A} b_{ij} - (C - \sum_{ij,\text{where } B_{ij}\notin A} b_{ij}) \\ &= \sum_{ij,\text{where } B_{ij}\in A} b_{ij} + \sum_{ij,\text{where } B_{ij}\notin A} b_{ij} - C \\ &= \sum_{ij,\forall i\in\{1,2,\ldots,n\},j\in\{1,2,\ldots,n\}} b_{ij} - C \end{aligned}$$

- $\sum_{ij,\forall i\in\{1,2,\ldots,n\},j\in\{1,2,\ldots,n\}} b_{ij}$ is a constant for this network, so we see now that by minimizing this cut, we maximize the profit. Our formulation of the flow network is therefore correct.

(Time complexity)

- There are $O(n^2)$ nodes.
- There are also $O(n^2)$ edges, but we expect the exact number of edges to be greater than the exact number of nodes.
- C can be arbitrarily large.
- Because we don't have a bound on C and the problem specifically asks for a polynomial-time algorithm, we don't want something that depends on C . We are thus limited to Edmonds-Karp and preflow-push. With our knowledge of the problem, we see that preflow-push scales better (albeit with the same complexity), so our total complexity is $O((n^2)^2n^2) = O(n^6)$.