CMSC 27200 - Problem Set 7

Sohini Banerjee

March 2, 2024

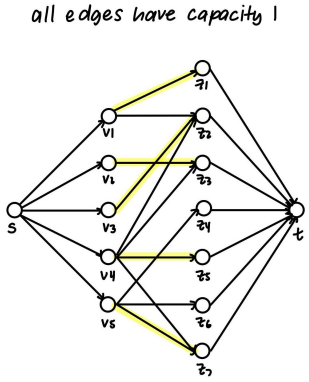
1a

Verifier

This problem is in NP. As stated in Ed, if an article belongs to multiple subgroups, we count it as part of one of these subgroups only and choose which group to count it in.

Let the certificate be a sequence of m vertices $X \subseteq V$ where $X = [v_1, ..., v_m]$. Below is a polynomial time verifier algorithm:

1. Check that X forms a legitimate path.
   (a) Check that there exists an edge from v_i to v_{i+1} for $i \in [1, m-1]$. If there are vertices v_i and v_{i+1} such that there is no edge from v_i to v_{i+1}, output NO.
2. Check that $v_1 = s$ and $v_m = t$. If not, output NO.
3. Use Ford-Fulkerson to check that for each subset Z_j for $j \in [1, k]$, there is at most 1 vertex $x \in V$ such that $x \in Z_j$.
   (a) Create a new graph G' (a sample graph is shown below) by doing the following:
      i. Create a source vertex s and sink vertex t.
      ii. Create a vertex for every v_i in the sequence X.
      iii. Create a vertex for every subgroup Z_j.
      iv. Add an edge from s to each v_i.
      v. Add an edge from v_i to each Z_j such that article v_i belongs to subgroup Z_j.
      vi. Add an edge from each Z_j to t.
      vii. Let all the edges have capacity 1.
   (b) Run Ford-Fulkerson to find the maximum flow from s to t. If the maximum flow is m, output YES. Otherwise, output NO.



Explanation

In step 1, the verifier iterates through each vertex and checks that there exists an edge between adjacent vertices in the given sequence. If there is a missing edge, then the given path is not a valid path, so it cannot be an $s-t$ path, so we immediately output NO. For each vertex, we check whether there exists an edge to the next vertex, which takes $O(|E|)$ time, and repeating this for every vertex takes $O(|V||E|)$ time.

In step 2, the verifier ensures that the given path is an $s-t$ path. That is, it starts at s and ends on t. This takes $O(1)$ time.

In step 3, the verifier checks the subgroup constraint. Since there is a capacity of 1 for an edge exiting Z_j, the flow entering Z_j can be at most 1, so only 1 article is counted as part of that subgroup. If we reach a flow of m, it means that all m articles passed through a subgroup and no subgroup had multiple articles passing through them, so we found a valid $s-t$ path that meets the subgroup constraint and output YES. Otherwise, the constraint is not met, so we output NO. The maximum flow is bounded by m since there are m edges leaving s each with a capacity of 1, and m is bounded by $|V|$. Furthermore, there are $|V|k$ possible edges, one from each article to a subgroup. Therefore, Ford-Fulkerson takes $O(|V|^2k)$ time, which is polynomial.

1a (Extra Credit)

This problem asks whether the problem is in NP given a function $Test(v, i)$ that returns 1 if vertex v (an article) is in subgroup Z_i and 0 otherwise, for some very large k. This problem is not in NP because to verify that there are not two articles in our sequence of vertices X that appear in the same group, we would have to iterate through all subgroups Z_i and run $Test(v, i)$ to determine what subgroup v is in, which is not polynomial if there are an exponential number of subgroups.

This is made more complicated by the fact that we can choose what subgroup to count each article for (if it appears in multiple subgroups), so running Ford-Fulkerson in the verifier would be exponential. This is because there are k vertices entering t in G', so if k is exponential, we increase the number of edges in G' exponentially, which significantly increases the running time of Ford-Fulkerson and makes it not polynomial.

1b

This problem is NP. In fact, we will give a polynomial time algorithm to solve it!

Algorithm

We will check that there are at least $k+1$ edge-disjoint paths between every pair of vertices.

- Stored Data: Modify G to get G' by replacing each undirected edge $e = (u, v)$ with a directed edge of capacity 1 from u to v and another directed edge of capacity 1 from v to u.
- Initialization: Let $x = \infty$ be the minimum maximum flow between any two vertices.
- Iterative Step:
    1. For each pair of vertices u and v:
        (a) Run Ford-Fulkerson with u as the source and v as the sink and find the maximum flow from u to v, called y. Note that for v_1 and v_2, we run the algorithm 2 times, once with v_1 as the source and v_2 as the source.
        (b) Set $x = \min(x, y)$.
    2. If $y \geq k+1$, output YES. Otherwise, output NO.

Explanation

We know that all problems in P are in NP since we replace the verifier with the algorithm to solve the problem. We now explain why our algorithm is correct.

We first create G', which takes $O(|E|)$ time as we replace each undirected edge with 2 directed edges.

Our algorithm checks that that there are at least $k+1$ edge-disjoint paths between every pair of vertices. This ensures that even if we remove at most k edges, one from each path, there is still a path left connecting u and v. In other words, if we remove any subset of k edges, our graph remains connected.

There are $O(|V|^2)$ pairs of vertices to check. For each pair, we run Ford-Fulkerson, where the maximum flow is bounded by $|V|$ since there are a maximum of $|V|$ edges leaving the source u, one to each other vertex, which forms a cut of size $|V|$. Furthermore, there is a maximum of $|V|^2$ edges, such as if we had an edge between every pair of vertices. Therefore, since there are $|V|^2$ edges, and the maximum flow is bounded by $|V|$, each iteration of Ford-Fulkerson takes $O(|V|^3)$ time. Repeating this for every pair of vertices, the algorithm has a time complexity of $O(|V|^5)$, which is polynomial time. Therefore, this problem is in P, and since $P \subseteq NP$, this problem is in NP.

1c

Verifier

This problem is in NP.

Let the certificate be a set of vertices $L \subseteq V$. Below is a polynomial time verifier algorithm:

- Let $R = V(G) - L$ be the set of vertices in G that are not in L.
- For each edge $e = (u, v) \in E$, determine whether it satisfies one of the following: either $u \in L$ and $v \in R$ or $v \in L$ and $u \in R$. If the number of such edges is k, output YES. Otherwise, output NO.

Explanation

If the given solution has exactly k edges that cross the cut, we have found a cut that cuts exactly k edges so we output YES, as the answer to the problem is YES. If the given solution does not have exactly k edges that cross the cut, we output NO, as there may or may not be some other cut that cuts exactly k edges.

The verifier iterates through each edge and determines whether it cross the cut provided. For each edge, it takes $O(|V|)$ to determine whether $u \in L$ (if not, $u \in R$) and $v \in L$ (if not, $v \in R$). Therefore, the verifier has a time complexity of $O(|E||V|)$, which is polynomial time.

1d

This problem is not in NP.

Let the certificate be a set of vertices $X \subseteq V$. The natural verifer of this problem would be, for every pair of vertices u and v, check whether there is an edge connecting u and v. While this can be done in polynomial time, iterating through every pair of vertices in $O(|V|^2)$ and checking whether an edge connects the two vertices in $O(|E|)$, making the overall time complexity $O(|E||V|^2)$, this does not solve the problem. The problem asks whether the maximum clique has a size of k. To do so, we would not only have to verify that our clique has a size $\leq k$, but also that no other clique has a size $\geq k+1$. To do so, we would have to check every set of $k+1$ vertices to make sure no clique of size $k+1$ exists, which is difficult to do in polynomial time. Therefore, this problem is not in NP.

2

Algorithm

Let $M(i, j)$ be the maximum number of points we can get by landing on position i in exactly j jumps. Let $P(i, j)$ be the previous position we hopped from to land on position i in exactly k jumps.

- Base Cases:
  - $M(i, j) = -\infty$ for $i < 0$ and $j \in [0, k]$ since we cannot accumulate points at non-positive positions.
  - $M(i, j) = 0$ for $i = 0$ and $j \in [0, k]$ since we have no points when starting.
  - $M(i, 0) = -\infty$ for $i \in [1, n]$ since we cannot reach a positive position without a jump.
  - $P(i, j) = \emptyset$ for $i \leq 0$ and $j \in [0, k]$ since non-positive positions cannot have had a previous position.
- Recurrence Relation: $M(i, j) = \max\{M(i - k, j - 1) + v_i\}$ for $k \in [1, 5]$
  - Note that if $M(i - k, j - 1) = -\infty$, we define $M(i - k, j - 1) + v_i = -\infty$ as well. In other words, if we are hopping from some position $i - k$ that cannot be reached in $j - 1$ hops, we cannot hop from position $i - k$ to i in j hops.
  - Suppose hopping from position $i - k'$ using $j - 1$ hops to position i using j hops maximizes points where $M(i - k', j - 1) \neq -\infty$. Then, set $P(i, j) = i - k'$.
- Solution: The maximum points is given by $\max(M(n, j))$ for $j \in [1, k]$. We can retrieve the sequence of jumps by doing the following:
  - Suppose reaching position n with k' jumps maximizes points at position n. Then, the previous position is given by $i' = P(n, k')$. From this position, suppose reaching position i' with k^* points maximizes points at position i'. Then, the previous position is given by $i^* = P(i', k^*)$. We repeat this procedure until $i' = 0$, meaning we reached the start of the path. We then reverse the positions visited to get our sequence of jumps.

Proof

- First, we will show that $M(i, j) \geq \max\{M(i-k, j-1)+v_i\}$ for $k \in [1, 5]$. There are 2 cases to consider:
  - $M(i - k, j - 1) = -\infty$: In this case, $M(i, j) = -\infty$, so we are done.
  - $M(i-k, j-1) \neq -\infty$: By definition of $M(i-k, j-1)$ there exists some series of hops S that reaches position $i - k$ using $j - 1$ hops. Adding a hop of size k gives us a series of hops S' that reaches position i using j hops and has value $M(i-k, j-1)+v_i$. Thus, we have $M(i, j) \geq M(i-k, j-1)+v_i$.
- Second, we will show that $M(i, j) \leq \max\{M(i - k, j - 1) + v_i\}$ for $k \in [1, 5]$. There are 2 cases to consider:
  - $M(i - k, j - 1) = -\infty$. In this case, $M(i, j) = -\infty$, so we are done.
  - $M(i - k, j - 1) \neq -\infty$: By definition of $M(i, j)$, there exists some series of hops S' that reaches position i using j hops. Since we must have jumped from a previous position, S' consists of a series of hops S that reaches position $i - k$ in $j - 1$ jumps, plus the hop of size k from $i - k$ to i for a total of j hops where gain v_i in value. $M(S) \leq M(i - k, j - 1)$, so we have that $M(i, j) = M(S') = M(S) + v_i \leq M(i - k, j - 1) + v_i$. Thus, we have that $M(i, j) \leq M(i - k, j - 1) + v_i$.

Taken together, we get that $M(i, j) = \max\{M(i - k, j - 1) + v_i\}$ for $k \in [1, 5]$. Furthermore, since we must have taken some number of hops to reach position n, we check the maximum points at position n for all numbers of total hops from 0 to n.

Runtime

There are nk subproblems to solve for each combination of position and number of hops. Solving each problem takes $O(1)$ since we check $j-1$ hops at 5 different positions. Thus, the overall time complexity is $O(nk)$, which is polynomial time.

3

Algorithm

We can reduce this problem to a maximum flow problem by creating the graph G:

- Create 2 vertices v_{in} and v_{out} to represent each (x, y) vertex.
- Create a source vertex s and sink vertex t.
- Suppose $v = (x, y)$ is a starting vertex. Add an edge from s to v_{in}.
- Suppose $v = (x, y)$ is a boundary vertex. Add an edge from v_{out} to t.
- Suppose $v = (x, y)$ and $v' = (x', y')$ are adjacent vertices. Add an edge from v_{out} to v'_{in} and another edge from v'_{out} to v_{in}.
- Let all the edges created have a capacity of 1.

Then, we run Ford-Fulkerson to find the maximum flow from s to t. If the flow is m, output YES. Otherwise, output NO.

Proof

We need to show that there are m vertex-disjoint paths from the starting points to the boundary if and only if there is a flow of value m on G.

Part 1
First, we will show that if there are m vertex-disjoint paths from each starting point to the boundary, then there is a flow of value m on G.

For each path $P = v_1 \to v_2 \to ... \to v_l$, we can route flow on G:

- 1 unit from s to $v_{1_{in}}$
- 1 unit from $v_{l_{out}}$ to t
- 1 unit from $v_{i_{in}}$ to $v_{i_{out}}$ for $i \in [l]$
- 1 unit from $v_{i_{out}}$ to $v_{i+1_{in}}$ for $i \in [1, l-1]$

We start by routing 1 unit of flow from s to $v_{1_{in}}$ for each path P, so the flow leaving s is m. Then, we route 1 unit of along along each vertex in P, passing through the in and out vertex for each vertex in P. In other words, we route the flow like this: $s \to v_{1_{in}} \to v_{1_{out}} \to ... \to v_{l_{in}} \to v_{l_{out}} \to t$. So, we have 1 unit of flow from $v_{l_{out}}$ for each path (where l depends on the length of P) to t, so the flow entering t is m, since there are m vertex-disjoint paths. Since we have m vertex-disjoint paths, any vertex v is visited at most once, so the flow through v_{in} and v_{out} cannot exceed its capacity of 1. Since we have m units of flow leaving s and entering t and all edge capacities are satisfied, we have a flow of value m on G.

Part 2
Second, we will show that given an integer-valued flow of m on G, there are m vertex-disjoint paths from each starting point to the boundary.

An integer valued flow means there are m units of flow exiting s and m units of flow entering t. Furthermore, there is 1 unit from s to $v_{i_{in}}$ for some vertex v_i for m such vertices. From every such vertex, it must be routed to its out vertex, with capacity 1, so there is at most 1 unit of flow through each vertex v_i. Therefore, starting at $v_{1_{in}}$ for some v_1 that receives a unit of flow from s, we can find a path from s to t with a structure of $s \to v_{1_{in}} \to v_{1_{out}} \to ... \to v_{l_{in}} \to v_{l_{out}} \to t$. Therefore, we can construct a path $P = v_1 \to v_2 \to ... \to v_l$ for every such path that receives a unit of flow from s. Since the flow entering t is m, there are m such paths we can construct, which are vertex-disjoint.

Furthermore, we can verify that for any original edge $e = (u, v)$, creating an edge from v_{out} to v'_{in} and another edge from v'_{out} to v_{in} will not violate the vertex-disjoint propery. This is for 2 reasons:

- Ford-Fulkerson does not care whether we have a cycle and adding in and out edges only changes the flow in that each vertex is used at most once, regardless of the remaining structure of the graph.
- More formally, for any vertex u to be used in multiple paths, it must appear as $u_{in} \to u_{out}$ in both paths, since we created no edge for flow to be routed from out to in. However, the in to out edge has a capacity of 1, which prevents multiple paths from using it.

Runtime

We run Ford-Fulkerson on G. The cut passing through the edges from s to v_{in} where v is a starting vertex has a value of m, since there m starting vertices. Furthermore, m is bounded by n^2, so there are a maximum of n^2 starting vertices. Each vertex has a maximum of 8 edges attached to it, since it can have 4 adjacent vertices and there is a forward and backward edge. However, this is a constant number of edges for each vertex, so the number of edges is bounded by $O(n^2)$. Therefore, running 1 iteration of Ford-Fulkerson takes $O(|E|) = O(n^2)$ time. Thus, the overall time complexity is $O(n^4)$, which is polynomial time.

4

Verifier

This problem is in NP.

We can let the certificate be the set of toppings T. Suppose there are M total toppings. Our verifier is that for each person, iterate through the toppings to ensure that at least one preferred topping is in T and no disliked topping is in T. If this condition holds for at least k people, output YES. Otherwise, output NO. This can be done easily in linear time, so we have a polynomial time verifier.

Explanation

We can verify each person's preferences in $O(M)$ time and doing this for each person makes the overall time complexity $O(nM)$, which is polynomial time. If we find that at least k people are happy, then the we output YES as the answer to the problem is YES. Otherwise, we output NO, as there may or may not be a set of toppings that satisfy at least k people.

Reduction

If we reduce an NP-complete problem to our problem, then our problem must also be NP-complete. Define the independent set problem as whether there exists an independent set of size at least k. We will reduce independent set to our problem by doing the following:

- Let the toppings be the set of all vertices.
- For each vertex $v_i \in V$, do the following:

- Create a person whose favorite toppings are v_i and disliked toppings are the set of v_j such that there exists an edge from v_i to v_j.

- Determine whether there is a pizza such that at least k people will eat.

We will show that there exists an independent size of k if and only if there is a pizza that k people will eat. Note the following: No two people like the same topping, based on the construction above, since a topping is only liked when we create a set of likes and dislikes for a single vertex (and the liked topping is this vertex).

Part 1
First, we will show that if there is a pizza that k people will eat, then there is an independent set of size k. Suppose that we take the k people that eat this pizza. We know they all have different liked toppings (and only 1 liked topping each), so these k toppings must be in the pizza. Furthermore, none of the disliked toppings for these people appear in the pizza. Since each topping represents a vertex, and the disliked toppings of the person whose liked topping is v_i are not in the pizza, we know the adjacent vertices to v_i are not in the pizza since those are the disliked toppings. Therefore, using the toppings in the pizza, we can construct an independent set of size k.

Part 2
Second, we will show that if there is an independent set of size k, then there is a pizza that k people will eat. Suppose we take an independent set V of size k. Then, for each vertex $v_i \in V$, none of the adjacent vertices are in the independent set. By our definition above, each vertex v_i is the favorite topping of only 1 person. If we put v_i on the pizza, the person who liked v_i will eat the pizza if all its disliked toppings are not on the pizza. This will be the case because its disliked toppings are adjacent vertices, which cannot be in V, so we would not place them on the pizza. Therefore, each person whose favorite topping is $v_i \in V$ (there are k people like this) will all be satisfied because their disliked toppings, vertices adjacent to v_i will not be a selected topping by definition of independent set.