CSCI 270 Problem Set 8 Solutions

Fall 2025

Problem 1. *We studied the following lemmas in the lecture. Please complete the detailed proofs.*

Lemma 0.1. *Let f be a flow on a graph G, and let $(S, \overline{S})$ a s-t cut. Then*

$$v(f) = \sum_{e=(u,v):u\in S, v\notin S} f(e) - \sum_{e=(u,v):u\notin S, v\in S} f(e).$$

Here $v(f)$ is the flow value of f, defined as the flowing exiting s, i.e., $v(f) = \sum_{e=(s,v):v\in V} f(e)$. Recall that we assume s does not have any edges pointing towards it in G, for simplicity.

Lemma 0.2. *Let f be a flow, and assume that the flow value on each edge is an integer. Then f can be decomposed into a combination of path flows.*

Solution. **Lemma 0.1.** Consider the quantity

$$\Phi(S) \coloneqq \sum_{u\in S}\left(\sum_{(u,w)\in E} f(u,w) \; - \sum_{(z,u)\in E} f(z,u)\right).$$

By flow conservation, each $u \in S \setminus \{s\}$ contributes zero to this sum. Since s has no incoming edges, the net outflow from s is exactly $v(f)$, and hence $\Phi(S) = v(f)$. To compute $\Phi(S)$ edge-by-edge, note that any edge (x, y) with $x, y \in S$ *contributes* $f(x, y)$ to the sum $\Phi(S)$ when $u = x$ is processed in the outermost sum, and *subtracts* $f(x, y)$ when $u = y$ is processed in the outermost sum. Thus it cancels to zero. Edges with $x, y \notin S$ likewise contribute nothing. An edge with $x \in S$ and $y \notin S$ contributes $+f(x, y)$, and an edge with $x \notin S$ and $y \in S$ contributes $-f(x, y)$. Therefore

$$\Phi(S) = \sum_{e=(u,v):u\in S, v\notin S} f(e) - \sum_{e=(u,v):u\notin S, v\in S} f(e).$$

Equating the two expressions for $\Phi(S)$ yields the desired identity.

Lemma 0.2. We assume for simplicity that the graph G is acyclic. Let $f^{(0)} = f$. At step k, if $f^{(k)}$ is the zero flow, terminate. Otherwise consider the directed subgraph $G^{(k)} = (V, E^{(k)})$ where $E^{(k)} = \{e : f^{(k)}(e) > 0\}$. Since G is acyclic, any positive s–t flow must induce at least one directed s–t path in $G^{(k)}$. Select such a path $P^{(k)}$. Define

$$\Delta_k \coloneqq \min_{e\in P^{(k)}} f^{(k)}(e),$$

which is a positive integer. Define the path flow $p^{(k)}$ supported on $P^{(k)}$ by $p^{(k)}(e) = \Delta_k$ for $e \in P^{(k)}$ and 0 otherwise. Then set $f^{(k+1)} \coloneqq f^{(k)} - p^{(k)}$. This is a feasible flow because subtracting

a path flow preserves flow conservation at all intermediate vertices, and $p^{(k)}(e) \leq f^{(k)}(e)$ for all edges. At least one edge in $P^{(k)}$ attains the minimum Δ_k, so its flow becomes 0 in $f^{(k+1)}$. Since all values $f^{(k)}(e)$ are integers, the multi-set of positive values strictly decreases at each iteration, ensuring termination after finitely many steps. Summing over all steps gives

$$f = \sum_{i \in I} p^{(i)}.$$

Each $p^{(i)}$ is an s–t path flow with integer value, establishing the decomposition. □

Problem 2. *USC plans to offer m courses in Spring 2026. Each class i has an enrollment cap c_i. Suppose that USC has n students. Each student j has a list of classes that they want to take at some point, as well as a number k_j of classes they would like to take next semester. Here, we assume that k_j is smaller than the total number of courses.*

The USC Academic Affairs Office now begins assigning classes to students. They want to know whether, given the preference lists and class capacities, it is possible to assign each student the number of classes they want. Give and analyze a polynomial-time algorithm which decides whether this is possible, and outputs a class assignment satisfying all constraints if one exists.

This solution does a great job of motivating the solution, but you definitely don't need to be this wordy in an exam setting! You can concisely state the correct reduction, analyze its runtime, and prove its correctness.

Proof. This problem is very similar to a bipartite matching problem, except that students must be matched multiple times, and classes can be matched multiple times. As a result, we can use a very similar reduction to integer max-flow as we did for the matching problem in bipartite graphs. Before doing so, let's quickly explore a reduction to matching — reductions of that type often work, but here, there's a subtle problem.

We could create, for each student j, k_j copies of this student, and each of these copies now wants to take one class. Similarly, for each class i, we can create c_i copies of that class, and each now has capacity only one. For the bipartite graph, we create an edge between a copy of j and a copy of i if and only if there was an edge between j and i in the original input. Now, we could test whether there is a bipartite matching in which all copies of all students are matched. This almost works: the one problem is that we might have multiple copies of the same student be matched with multiple copies of the same class, e.g., a student meets their target of 4 classes by enrolling in CSCI 270 four times. It's not that clear how to fix this directly.[1] You should look at how our upcoming reduction avoids this problem.

For our reduction to maximum s-t flow, we do something very similar to what we did for bipartite matching. We build a flow network G. We have one node for each student j,

[1] Also, this reduction could take pseudo-polynomial time if you have very large k_j and c_i values, whereas a direct solution would typically work in polynomial time.

one node for each class i, a source s and a sink t. From the source, we have an edge (s, j) to each student j, and give it a capacity of k_j — this ensures that no student will ever take more than k_j classes. For each class i, we have an edge of capacity c_i to the sink t, ensuring that no class ever has more than c_i students in it. Finally, whenever a student j is interested in class i, we add an edge from j to i of capacity 1. Different from our bipartite matching reduction, the capacity of 1 is important here. If we made the capacity larger, we would run into the problem from the previous paragraph: a student might send more than one unit of flow to a particular class, which we would have to interpret as that student enrolling in the class multiple times. A capacity of 1 prevents this.

After building G with the capacities, we run the Edmonds-Karp Algorithm[2] on this input to compute a maximum integer s-t flow f. If the value of this flow is $\sum_j k_j$ (i.e., the total number of classes all students want to take), we output the assignment which gives each student j all classes i such that $f(j, i) = 1$. If the value of the flow is less, then we output that there is no possible assignment. This algorithm takes polynomial time, because the pre-processing is very direct, and Edmonds-Karp takes strongly polynomial time.[3] And the post-processing of looking which edges have flow 1 is also just $O(nm)$.

We now prove correctness. First, we show that if the algorithm says that there is no way to give each student the number of classes they want, then there is in fact no way. We prove this by contrapositive: assume that there is a way to give each student k_j classes out of the one they are interested in. In that case, for each j, let S_j be the set of $|S_j| = k_j$ classes that make it possible. Consider the flow which, for each j and each $i \in S_j$, routes one unit from s to j to i to t. Conservation is guaranteed because we defined the flow via paths. On the middle edges (j, i), capacity is respected because we had a *set* of classes for each student, so each i only appeared once. On the edges from s to j, because the student takes k_j classes in the solution, we route exactly k_j units of flow. And on the edges from i to t, we route at most c_i units, because the given solution did not exceed capacity of any class. So the flow is valid. Because it sends k_j units on each edge (s, j), the value of the flow is $\sum_j k_j$, so we have shown that such a flow exists. The Edmonds-Karp algorithm finds a maximum flow, so it must also have achieved this value, and output something.

For the converse direction, we assume that the algorithm outputs an assignment of students to classes. For each student j, define S_j to be the set of classes the algorithm gave the student. Because the algorithm used an integer maximum s-t flow f, each edge from some j to some i carried either 0 or 1 unit of flow. Because the value of the flow was $\sum_j k_j$ in this case (the algorithm output something), all the edges out of s must have been fully saturated, i.e., had flow k_j. By conservation, for each student j, we chose exactly k_j classes, so $|S_j| = k_j$. This uses that each edge from j to i can only carry one unit of flow because it had capacity 1. Finally, no class exceeds its capacity. This is because the total flow into node i is the total number of students j with $i \in S_j$, i.e., who are assigned class i. By conservation, this must equal the flow out of i. And because the only edge out of i is the one to t with capacity c_i,

[2]The Edmonds-Karp algorithm is the version of Ford-Fulkerson that in each iteration chooses the *shortest* s-t path in the residual graph. It has runtime $O(VE^2)$, whereas Ford-Fulkerson can technically have exponential runtime with adversarial path choices.

[3]Ford-Fulkerson, i.e., no specific choice of path, would be ok here as well. First, we said that the k_j are "typically" smaller than the number of classes the student is interested in. But even if they were really large, notice that the cut between all students j and all classes i cuts at most mn edges, which is an upper bound on the total value of the maximum s-t flow. So here, Ford-Fulkerson would take at most mn iterations.

and f did not violate that capacity, the number of students assigned to class i is at most c_i. This completes the correctness proof.

□