(Sol) HW08 Max-Flow vs. Min-Cut

Due: Monday, November 1st, 2021 11:59pm on Canvas

KT 7.4~7.10

Duplicate, edit and export as a pdf to submit on Canvas. No late submission is accepted.

1. [12 points] Classroom Assignment

   We have n courses (each course runs for 1 hour) and we have m classrooms. a_i students are in a course i, and course i runs from A_i hours to $A_i + 1$ hours (e.g., from 10:00 AM to 11:00 AM or from noon to 1 PM). A classroom j can hold up to d_j students, and it can be used between B_j and C_j hours (e.g., between 10:00 AM and 4:00 PM). Of course, each classroom can host at most one course at a time (but it can host many courses back-to-back).

   Let us assume that if a class ends at 10:00AM and another class begins at 10:00AM, then these two do not conflict (hence the same classroom can host those two courses back-to-back).

   Given $n, m, \{a_i, A_i\}, \{d_j, B_j, C_j\}$ as input, we want to assign as many courses to classrooms (without conflicts) as possible.

   Design an efficient algorithm that can find a maximum assignment (the cardinality of an assignment is the number of courses assigned to classroom).

   (a) [6 points] Describe your reduction. This means, you need to clearly describe the following:

   - Construction - How you would construct a bipartite graph G from an arbitrary input instance $n, m, \{a_i, A_i\}, \{d_j, B_j, C_j\}$.
     - For every course i and classroom j, if $a_i \leq d_j$, $B_j \leq A_i$, and $A_i + 1 \leq C_j$, add an edge $(i, j@A_i)$ from vertex i to vertex $j@A_i$. (There will be $|C_j - B_j|$ vertices for the same classroom j.)

- Conversion - How you would convert a bipartite matching in G into an assignment.
    - Given the matching M, for every edge $(i, j@t)$ in M, assign course i to classroom j at time t.
- Relationship - What is the relationship between the size of an assignment and the value of a matching in your reduction? (e.g., are they equal? is one twice the other? etc.)
    - The size of assignment, i.e. how many courses are scheduled to a classroom, is equal to the size of matching M.

(b) [6 points] Analyze running time of your reduction. This involves three steps:

- Running time of the Construction step (express your answer in terms of $n, m, \{a_i, A_i\}, \{d_j, B_j, C_j\}$ using Big-O).
    - For each pair of course i and classroom j, the constant amount of work is required, so the overall running time is $O(nm)$.
- Running time of the Conversion step (express your answer in terms of $n, m, \{a_i, A_i\}, \{d_j, B_j, C_j\}$ using Big-O).
    - The conversion goes through every edge in the matching, and the size of matching cannot be larger than the number of courses n, so $O(n)$.
- Running time of finding a max matching in G (express your answer in terms of $n, m, \{a_i, A_i\}, \{d_j, B_j, C_j\}$ using Big-O).
    - Ford-Fulkerson algorithm runs in $O(c|V||E|)$, where c is the max edge capacity.
    - Once the max flow is found, we need to go through every edge to find the max matching, which takes $O(|E|)$.
    - The number of vertices in this flow graph $|V|$ is $n + m + 2$, and the number of edges in this flow graph $|E|$ is at most $(n + 24m + n * 24m)$. c is 1. Thus the overall time complexity is $O((n + m + 2) \times (n + 24m + 24mn))$, or $O(n^2m)$. (The number of classroom is likely to be smaller than the number of courses, so n dominates m.)

2. [8 points] Profit Maximization

You are selling customized laptops with pre-installed apps.

There are n apps ($a_1, a_2, \cdots, a_n$), and you need to pay for license fees ($f_1, f_2, \cdots, f_n > 0$) to pre-install those apps on customized laptops. Specifically, if you pay f_i for an app a_i, then you are allowed to pre-install it on as many laptops as you want (there is no limit).

You have m customers, and they want different sets of pre-installed apps on their laptop. Customer j will buy a laptop from you and pay r_j > 0 dollars if you can pre-install the apps on their laptop, according to W_j ($W_j \in \{a_1, a_2, \cdots, a_n\}$). Each customer will buy at most one laptop from you.

You have m laptops (with no pre-installed apps yet), and you need to decide which app licenses you need to purchase in order to maximize your net profit. In some cases, your maximum profit may be 0 if it's better not to purchase any licenses (and thus to sell no laptops).

Examples:

Suppose $n = 3$ with $f_1 = 30$, $f_2 = 10$, $f_3 = 200$ and $m = 2$, $W_1 = \{a_1, a_2\}$, $W_2 = \{a_1, a_3\}$, $r_1 = 45$, and $r_2 = 235$.

- If you sell no laptops, your profit will be 0 dollars.
- If you want to sell a laptop to customer 1, then you need to pay $f_1 + f_2$ for the licenses; the profit will be $r_1 - (f_1 + f_2) = 5$ dollars.
- If you want to sell a laptop to customer 2, then you need to pay $f_1 + f_3$ for the licenses; the profit will be $r_2 - (f_1 + f_3) = 5$ dollars.
- If you want to sell a laptop to both customers, then you need to pay $f_1 + f_2 + f_3$ for the licenses (note that you don't pay twice for the license of a_1); the profit will be 45 + 235 − (30 + 10 + 200) = 40. This is optimal.

Suppose $n = 3$ with $f_1 = 10$, $f_2 = 10$, $f_3 = 12$ and $m = 2$, $W_1 = \{a_1, a_2\}$, $W_2 = \{a_1, a_3\}$, $r_1 = 15$, and $r_2 = 10$.

- If you sell no laptops, your profit will be 0 dollars. (This is optimal.)

- If you want to sell a laptop to customer 1, then you need to pay $f_1 + f_2$ for the licenses; the profit will be $r_1 - (f_1 + f_2) = -5$ dollars.
- If you want to sell a laptop to customer 2, then you need to pay $f_1 + f_3$ for the licenses; the profit will be $r_2 - (f_1 + f_3) = -12$ dollars.
- If you want to sell a laptop to each customer, then you need to pay $f_1 + f_2 + f_3$ for the licenses (note that you don't pay twice for the license of a1); the profit will be 15 + 10 − (10 + 10 + 12) = -7 dollars.

To describe a solution, you can simply list a set of customers to whom you'll sell laptops (then, it's clear that you'll purchase the necessary licenses only, to maximize your profit).

This problem can be solved by reducing it to the min-cut problem.

(a) [0 points] Consider the following example. Find an optimal solution and describe to which customer(s) you'll sell laptops. Suppose $n = 3$ with $f_1 = 10$, $f_2 = 25$, $f_3 = 12$ and $m = 3$, $W_1 = \{a_1, a_2\}$, $W_2 = \{a_1, a_3\}$, $W_3 = \{a_2, a_3\}$, $r_1 = 10$, $r_2 = 55$, $r_3 = 12$.

The optimal solution is to sell the second laptop, $r_2 - f_1 - f_3 = 33$.

(b) [2 points] Describe how you can construct a flow network G (given $n, m, \{f_i\}, \{W_j, r_j\}$) in polynomial time. You should describe how you create/define nodes and edges (and capacities) of G, in terms of $n, m, \{f_i\}, \{W_j, r_j\}$.
Hint:.{t,s} sulp, remotsuc hcae rof edon eno dna ppa hcae rof edon eno tnaw lliw uoY
Hint2:.deriuqer ton tub ,∞ eb ot segde emos fo yticapac eht enifed ot lufesu eb yam tI

- Create s, t, customer nodes and app nodes.
- Create an edge from s to each customer i with r_i as capacity.
- Create an edge from each app a_j to t with f_j as capacity.
- Create an edge from customer i to app a_j if a_j is in W_i, with infinite capacity.

(c) [2 points] Describe how you can convert a max-flow f^* in G into a set of customers you'll sell laptops to (note that you will probably need to find a min-cut (A^*, B^*) first).

Given the max flow f^*, use BFS or DFS on the residual graph of G to find the set of reachable nodes from s. A^* = $\{s\} \cup$ the set of reachable nodes from s and $B^* = V \setminus A^*$.

Once you have the min-cut(A^*, B^*), then add every customer node in A^* to the set of customers you'll sell laptops to.

(d) [2 points] Explain why minimizing the capacity of a cut in G is equivalent to maximizing the profit in the original problem.

Assume that C is the set of customers you will sell laptops to.

The profit = $\sum_{i \in C} r_i - \sum_{a_j \in W_i, i \in C} f_j$

The capacity of a cut(A, B) = $\sum_{i \notin A} r_i$ + $\sum_{a_j \in A} f_j$. Note that any edge from customer i to app a_j, if a_j is in W_i, is within A as the capacity of (i, a_j) is infinity. In other words, $a_j \in A$ iff $a_j \in W_i$ and $i \in A$.

The capacity of a cut(A, B) = $\sum_{i \notin A} r_i$ + $\sum_{a_j \in W_i, i \in A} f_j$

= $\sum_i r_i$ for all customer i - $\sum_{i \in A} r_i + \sum_{a_j \in W_i, i \in A} f_j$

= $\sum_i r_i$ for all customer i - $(\sum_{i \in A} r_i - \sum_{a_j \in W_i, i \in A} f_j)$

Note that ($\sum_i r_i$ for all customer i) is constant, so minimizing the capacity of a cut(A, B) is to find that A that maximizes $\sum_{i \in A} r_i - \sum_{j \in W_i, i \in A} f_j$, which gives the maximum profit when $A = C$.

(e) [2 points] Also, analyze the running time of finding a min-cut and converting it into a set of customers you will sell laptops to.

Finding the max-flow f^*: Using Ford-Fulkerson, that would be $O(|E|C)$ = $O(nmC)$ where C = ($\sum_i r_i$ for all customer i). (Note that in $O(c|V||E|)$, $c|V|$ is the upper bound of the max flow value. In this problem, c is infinity so we can't use $O(c|V||E|)$ as a time complexity.)

Given the max flow f^*, the running time of building the residual graph is $O(|E|)$. The running time of BFS or DFS on a connected graph is $O(|E|)$ as well, so the running time of converting a max flow into a min-cut is $O(|E|)$ = $O(nm)$.

Given the set A^*, you simply need to add customer nodes in A^* to the set of customers you will sell the laptops to, so the running time is $|A^*|$. The size of set A^* cannot be larger than $(n + m + 1)$, so $O(n + m)$.