6.854 Advanced Algorithms
Problem Set 6

John Wang
Collaborators:

Problem 1-a: Assuming there is a feasible solution, show how the problem can be solved by an application of min-cost flow.

Solution: We will create a pseudo bipartite graph. On the right side, there will be the set P of prefrosh. On the left side, we will have the set S of students. We also have three additional layers in the flow graph. There will be a set of U of suites that are occupied by a student, a set F of all the floors that are occupied, and a set D of all the dormitories available.

For each one of these sets, we will represent a member x in the set with a node. We will also create two nodes s and t to serve as the source and sink nodes respectively.

From the source node s, we will have edges (s, d_i) leading to each dormitory $d_i \in D$, each with capacity M_d, the maximum allowable number of preforsh in dorm d_i according to the firecode.. Next, from each dormitory $d_i \in D$, we will have edges leading to floors $f_i \in F_d$ for each floor in dormitory d_i. These edges will be (d_i, f_i) and will have capacity M_{f_i} which is the total number of prefrosh on floor f_i according to the firecode. Next, we will have edges (f_i, u_i) from floor f_i to each suite $u_i \in U$ which is located on floor f_i. The capacity of these edges will be m_{u_i}, which is the maximum number of prefrosh available for suite u_i.

Now, we will have edges (u_i, s_i) from each suite u_i to the students s_i who are occupying that suite. The capacity of each (u_i, s_i) edge will be 1.Finally, each student s_i will have an edge to every prefrosh p_i, (s_i, p_i). These edges will have capacity of 1 and a cost of $-f(s_i, p_i)$ where f is the suitability function. Finally, each prefrosh's node p_i will be connected to the sink t with edges (p_i, t), each of capacity 1.

Edges whose costs have not been mentioned, which are all edges except for (s_i, p_i), will have costs of 0. Performing min-cost max-flow on this graph will producce an output which maximizes the total suitability subject to our constraints. If there is no feasible solution, then the flow into t will not be equal to the total number of prefrosh.

We can show this algorithm is correct, firstly, by noticing that each augmenting path will have a flow of 1, since the capacity is limited by 1 on multiple edges. This means we can maximize the total suitability and make sure that each prefrosh is matched up with exactly one student (since each student has incoming capacity of 1, so can have max flow of 1). Moreover, we know we have satisfied the other constraints, because each dorm, floor, and suite cannot have more prefrosh than the capacities flowing through their nodes, and their capacities are created by the constraints given in the problem. $\square$

Problem 1-b: If there is no feasible solution, it may be necessary to break the limits on suites (but the fire-code is inflexible). Give an efficient algorithm that finds the solution that minimizes the total overage (sum of amounts by which individual suite limits m_s are broken) and, among such solutions, maximizes the total suitability. Your algorithm should detect if there is no such solution.

Solution: First we will find out what the minimum total overage is using min-cost max-flow. We do this by recreating the network G from problem 1-a. However, we will have a modified version of G. In our new network, G', we will change the cost of flows from students s_i to prefrosh p_j on edge (s_i, p_j). Now, the cost of this edge will be 0, and it will still have a capacity of 1. We will also add new edges from each floor f_i to each suite s_j. These new edges (f_i, s_j) will carry a cost of 1 and infinite capacity, and will represent the overage in each room. Since these are the only non-zero edges in the graph, the min-cost max flow algorithm will try to send as much flow through the regular paths before it relies on overages to each room. If the total flow into the sink t is not equal to the number of prefrosh, then there is no solution and we stop. If there is a solution, then we continue on.

Summing up to the total flow through each of these overage edges, we can get the minimum overage that is possible. This will allow us to now construct a new graph G'' which will solve our actual problem. We go back to our graph G from problem 1-a, but we add two new overage nodes O_1 and O_2.

It has a single incoming edge, which starts from the source s and ends at O. This edge (s, O) has capacity equal to the minimum overage that we calculated above and has cost 0. There are outgoing edges from O to each student s_i in the graph. We will have edges (O, s_i) each with capacity 1 and cost 0. Everything else in G'' is the same as in G.

This means that performing a min-cost max-flow on G'' will maximize the total suitability with the minimum total overage. This is because everything in G'' not going through the overage node O is the same as in problem 1-a, however, we now have the ability to send flow to students through the overage node.

$\square$

6.854 Advanced Algorithms
Problem Set 6

John Wang
Collaborators:

Problem 3-a: Suppose that you have an optimal solution to some minimum cost circulation problem with integer costs and you then change one edge cost by one unit. Show how you can reoptimize the solution in $O(mn)$ time. Note that this is faster than solving the minimum-cost circulation problem from scratch.

Solution: Suppose that an edge has been changed by one unit. Then, we know that this edge can have its cost increased or decreased by a single unit. If the flow was optimal before (as we are assuming), then changing the cost by a single unit will only change a single edge in the residual graph. There will be at most a single edge (u, v) with a negative cost in the residual graph, since before we assumed the solution to be optimal so that all costs were non-negative. Therefore, if we have introduced a cycle of negative cost, then it must go through (u, v). Moreover, that negative cost cycle must have all edges with cost of 0.

This means that we can easily augment along this cycle. We know that (u, v) must be included, so all we need to do is find a path from v to u of zero cost edges. We can do this using a BFS from v. If the BFS does not return anything, then there is no cycle of negative cost in the residual graph, which means that the flow is still optimal, so there is no modification needed. If the BFS does return a negative cost cycle (after adding (u, v) to the path, then we can augment the flow along the cycle.

After augmentation, if the (u, v) edge has no more capacity left, then we are done because there are no more negative edges left in the residual graph, and hence, no more negative cycles left in the residual graph. If there still is capacity left, we can change the price function to make the new reduced cost feasible. Notice that $c(u, v)$, the total cost on edge (u, v), has changed by 1.

Notice that that the reduced cost could only have changed by 1 after the augmentation. If we find all paths leading to v and increase their price functions by 1, then we will have a feasible reduced cost. This is because the only edge which may have decreased its cost is (u, v), and we are sure to have increased $p(u)$ by 1. Since $\Delta c(u, v) = -1$ in the worst case, we only need to show that $\Delta p(v) \leq 0$ in order for $c_p(u, v) \geq 0$ (since we know the previous reduced cost was non-negative), and for us to have found a feasible reduced cost. Now if $p(v) \geq 0$, then we could have increased the flow from v to u. Therefore, we have created a new price function with has a feasible reduced cost, and therefore, we have found an optimal circulation. $\square$

Problem 3-b: Deduce a cost-scaling algorithm for minimum-cost flow (with integer costs) that makes $O(m \log C)$ where C is the maximum cost, calls to your solution in part a and prove its correctness.

Solution: We perform the scaling algorithm by starting out with a graph G where all edges are changed to have cost 0 and each vertex has a price function of 0. We first find a max-flow on this and proceed to the scaling step. During each scaling step, we will double the cost of each edge, and also the price function. After we have double everything, we will add the last bit from the cost to each edge. Since there are m edges, we will have to make m calls to the algorithm during each scaling step. When we add the last bit, we will change the cost by a magnitude of 1. We can either have the cost increase or decrease by 1 depending on the sign. Since we know from the last part of the problem that we can reoptimize a min-cost circulation when we change a single edge cost by one unit, then we are able to get a new min-cost circulation for our modified graph. The repeat this for all m edges. Then, we will perform $\log C$ scaling steps in order to add all of the bits onto the end.

After $\log C$ scaling steps have been performed, we will end up with costs and price functions which are exactly those for the actual min-cost flow circulation problem. This means that we can be solve our original problem in $O(m \log C)$ calls to the algorithm from part a.

To prove correctness, we note that doubling the edge costs and the price function at each vertex does not change the min-cost flow. This is because the min cost flow is still just a constant multiple of its previous value (namely where the constant is 2). This follows since when we double all edge costs and all price functions, then each incoming and outgoing edge on a vertex have their costs changed in a symmetric manner. Adding a single bit to the end of each cost successively will also keep the invariant that our circulation is minimum

at all times, for whatever graph that is given. This is because immediately after doubling, we know that the flow is minimum as argued above. Also, we know from our argument in part a that new minimum cost flow can be found with the algorithm in part a. Therefore, immediately after adding the bit to change the cost on some edge (u, v), we can get a minimum-cost flow immediately after we perform the algorithm. Our base case starts out with a max-flow (of zero edge costs), so we obviously will have a min-cost flow. Since our invariant is maintained at each step, we can be sured once we get to the last step with the actual costs we want to compute, that our result will still be a min-cost flow. $\square$

6.854 Advanced Algorithms
Problem Set 6

John Wang
Collaborators:

> **Problem 4-a:** Give the optimum given $c = (-1, 0, 0)$.

Solution: We want to perform the following, given c: $min - x_1$ such that $x_1 + x_2 \geq 1$, $x_1 + 2x_2 \leq 3$, and $x_1 \geq 0$. Basically, we want to maximize x_1 subject to these constraints. Rearranging the constraints, we want:

$$1 - x_2 \leq x_1 \leq 3 - 2x_2 \tag{1}$$

Thus, in order to maximize x_1, we want to make x_2 as small as possible. If we choose $x_2 = 0$, then we have the smallest possible x_2 (since we also have the constraint that $x_2 \geq 0$). This means that $3 - x_2$ is maximized, so that we can mazimize x_1. We can therefore pick $x_1 = 3$. Therefore, we have the solution: $(x_1, x_2, x_3) = (3, 0, 0)$. $\square$

> **Problem 4-b:** Give the optimum given $c = (0, 1, 0)$.

Solution: We want to minimize x_2, so if we use the constraints, we can derive the expression $1 + x_1 \leq x_2 \leq 3/2 - (1/2)x_1$. Since we are minimizing x_2 and we know that $x_1 \geq 0$, we should choose x_1 as close to zero as possible. This means we choose $x_1 = 0$, which means 1 is the lower bound on x_2. Therefore, we have the solution: $(x_1, x_2, x_3) = (0, 1, 0)$. $\square$

> **Problem 4-c:** Give the optimum given $c = (0, 0, -1)$.

Solution: We want to minimize $-x_3$ or alternatively maximize x_3. Since there are no constraints on x_3 other than $x_3 \geq 0$, we can choose values of x_1 and x_2 which satisfy the other constraints, and just choose x_3 as large as possible. Therefore, we can choose $x_3 = \infty$, along with any values of x_1 and x_2 which are feasible. This means we have the solution: $(x_1, x_2, x_3) = (1, 1, \infty)$. $\square$