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. □

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: We will create a new network based on the network G that we created in problem 1-a. We will do this by creating new overflow nodes for each suite $u_i \in U$. We will have overflow nodes o_i which correspond to the overflow seen in each suite u_i. We will change the original network G by creating an edge from the floor of suite u_i to that suite's overflow node o_i. This edge o_i will have infinite capacity and cost K which is larger than the magnitude of any suitability $|f(x, y)|$ for all x, y where $f(x, y)$ is finite. In other words we set $K > \max_{x,y} |f(x, y)| \text{s.t.} \exists M, f(x, y) < M$. Thus, we take the maximum over $f(x, y)$ where $f(x, y)$ is bounded, and we can set $K = \max f(x, y) + 1$ in order to satisfy the inequality.

Note that we can always find a K because we are looking over bounded values of $f(x, y)$, which means we can find the maximum bounded value in $O(m)$ since there are $O(m)$ pairs of x and y. The outgoing edges from each overflow node o_i will be exactly the outgoing edges from the corresponding suite u_i. We will create edges of capacity 1 to each student who resides in suite u_i, so that we have edges (o_i, s_i) of cost 0.

The rest of the network graph G will be left the same, and now we will run min-cost flow from s to t like before. If there is a total flow equal to the number of prefrosh, then we have found a solution, where the overflow nodes represent the overflow in each room. Otherwise, we have no solution.

To show the correctness of this algorithm, we note that the costs K of going to the overflow nodes are larger than any other cost on the graph. This implies that in the shortest augmenting paths algorithm, the overflow edges will be used last (since all other edge costs are 0). Thus, the min-cost algorithm will fill up the matchings first by attempting to use all the available rooms without overflowing them.

Thus, the min-cost flow will minimize the amount of total overflow, since minimum cost will be dominated by obtaining minimum overflow. It is clear that a path through an overflow node is more costly than a path through any other node, which means that by minimizing cost, the algorithm is also minimizing overflow. The negative suitability cost between students and prefrosh will make sure to maximize the suitability, given that the algorithm first minimizes the overflow. $\square$

6.854 Advanced Algorithms

Problem Set 6

John Wang
Collaborators:

Problem 2: Consider a unit-capacities network where each edge has cost 1. Does the shortest augmenting path algorithm (for max-flow, that just minimizes the number of edges on the path) or Dinic's normal blocking-flow algorithm compute a min-cost max-flow in the graph? Why or why not?

Solution: Yes, both the shortest augmenting path algorithm and the block-flow algorithm will compute a min-cost max-flow on the graph. I claim that there will never be a negative reduced-cost cycle in the residual graph under the shortest augmenting path algoritm. This is clearly true at the beginning of the algorithm, when there is no flow going through the graph. Now I will use induction to show that it is never true after the ith augmentation.

Suppose that the above holds true for the $i-1$st augmentation. We can use shortest-paths and a supernode s' to compute distances from s' to each vertex v. We will use the shortest-paths distance as the price function. Since we know that $d(v) \leq c(u,v) + d(u)$ from the triangle inequality, which, when reordered, shows that $0 \leq d(u) + c(u,v) - d(v)$. Therefore, we see that the reduced cost distances are non-negative. Now, when we augment in the residual graph, each edge will have a reduced cost of 0.

This shows that the price function is still feasible because there are no negative reduced cost edges added after the ith augmentation. Since there are no negative reduced cost edges, there can be no negative reduced cost cycles. This completes the inductive step.

Now, since this holds for all augmentations, we know that once the shortest augmenting path algorithm finishes, there will be no negative reduced cost cycles in the graph, which means we have found a min-cost max-flow. □

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. □

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: $(x_1, x_2, x_3) = (3, 0, 0)$. □

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: $(x_1, x_2, x_3) = (0, 1, 0)$. □

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: $(x_1, x_2, x_3) = (1, 1, \infty)$. □

6.854 Advanced Algorithms

Problem Set 6

John Wang
Collaborators:

Problem 5-a: Formulate a linear program for maximizing the amount of Yen you have at the end of trading.

Solution: First, we will create variables x_i which will represents the orders that are possible for client i. Thus, x_i will be the number of units of currency of a_i that we will trade with client i for currency b_i. Now, we know that we must have $0 \leq x_i \leq u_i$ by the fact that client i is willing to trade no more than u_i.

Moreover, we know that the total amount of money that one can have in currency $c_j \in C$, where C is the set of all currencies except for the dollar, is given by $\sum_{a_i=c_j} x_i \leq \sum_{b_i=c_j} r_i x_i$. This is because one cannot spend more money than is possible by performing all the trades and getting interest rate r_i on all of the trades.

For dollars, we must add D to the right side to account for the fact that we start off in dollars, so that $\sum_{a_i=d} x_i \leq \sum_{b_i=d} r_i x_i + D$ where d denotes currency in dollars. These conditions come from the fact that however many units of currency c_j there are to exchange, namely $\sum_{a_i=c_j} x_i$, must be less than the total amount of currency and interest that one can possibly receive in currency c_j, namely $\sum_{b_i=c_j} r_i x_i$.

Finally, we want to maximize $\sum_{b_i=Y} r_i x_i - \sum_{a_i=Y} x_i$ where Y is the set of all orders in Yen. This will allow us the maximize the amount of Yen at the end of the trading day because we look for however much Yen was received, $\sum_{b_i=Y} r_i x_i$, with the amount of yen that we traded away subtracted from this, $\sum_{a_i=Y} x_i$.

Therefore, the full linear program is as follows:

$$\max \sum_{b_i=Y} r_i x_i - \sum_{a_i=Y} x_i \tag{2}$$

$$\text{s.t.} \sum_{a_i=c_j} x_i \leq \sum_{b_i=c_j} r_i x_i \quad \forall c_j \in C \tag{3}$$

$$\sum_{a_i=d} x_i \leq \sum_{b_i=d} r_i x_i + D \tag{4}$$

$$0 \leq x_i \leq u_i \quad \forall i \tag{5}$$

□

Problem 5-b: Show that it is possible to carry out trades to achieve the objective of the linear program without ever borrowing currency. (Hint: there is a sense in which your solution can be made acyclic).

Solution: I will show that borrowing currency and creating a cycle in the linear program does not provide anyhing for the objective function. In essence, whenever currency c_j is borrowed, one must create a cycle from currency c_j back to c_j. Since we know that $\prod r_i < 1$ for any cycle, we know that it is impossible to make a profit by arbitrage.

This means that if we borrow some amount a of currency c_j, and perform trades from $a_i = c_j$ to currency $b_i = c_j$ at rate r_i, we know that we will eventually need to return a in currency c_j. Therefore, whenever we borrow currency, we must create a cycle. However, by creating a cycle, have $\prod r_i < 1$ during that cycle for all r_i along the cycle. This means that by borrowing currency, we strictly lose money. The other possibility is that we borrow a in currency and leave this amount a until it is collected at the end of the trading day. However, this does not improve the objective function. This means that each optimal solution which borrows money in currency c_j and leaves it in currency c_j for the entire day will have an optimal counterpart which does not borrow from currency c_j and leave it unused in c_j.

We have seen that borrowing and creating a cycle back to currency c_j will cause one to have strictly less money than if one did not create a cycle. This implies that no optimum solution can have a cycle, otherwise,

one could just remove the cycle and have a better solution, which is a contradiction. Therefore, if there does exist an optimal solution, there must exist an optimal solution which does not borrow any money. □

Problem 5-c: Show that there is a sequence of trades that will let you end the day with the optimal amount of Yen and no other currency except dollars.

Solution: We will basically create a new linear program that will solve the problem using only dollars. To do this, we will first find the optimal amount of Yen, and make sure that the amount of Yen from our new program is no less than this optimal. Thus, let us set $\max \sum_{b_i=Y} r_i x_i - \sum_{a_i=Y} x_i = O$ as the optimal amount of Yen.

Now, we will have a new linear program with all the same constraints as before, but with a new object function. Additionally, we add the constraint that $\sum_{b_i=Y} r_i x_i - \sum_{a_i=Y} x_i \geq O$. The objective function will be to minimize the amount of dollars that are left at the end of the trading day. Thus, we will have $\min \sum_{a_i=d} x_i - \sum_{b_i=d} r_i x_i$, since $\sum_{b_i=d} r_i x_i$ is the amount of a different currency converted into dollars, and $\sum_{a_i=d} x_i$ is the amount of dollars converted into some other currency.

If we maximize the amount of dollars we have left (alternatively, minimize the number of dollars that are converted), we will be able to achieve a solution where we end the day with only Yen and dollars. First, note that since we have the condition that the amount of yen at the end be greater than or equal to O, we will remove from our optimal solution any cycles, since we have shown in problem 5-b that cycles only decrease the amount of money available and make a strictly worse solution. Thus, we know that no optimal solution can have any cycles, which means there will be no cycles in the solution to our new linear program.

In addition, the optimal solution to this problem will have no paths ending in a currency other than Yen. Assume, by contradiction, that there exists such a path p of a dollars ending at currency c_j. Then we must have achieved $\sum_{b_i=Y} r_i x_i - \sum_{a_i=Y} x_i \geq O$ so that we have at least an amount of yen greater than or equal to O. Since path p ends in currency c_j and not in Yen, path p is not contributing to the number of Yen at the end of the trading day. Therefore, we can remove it and still satisfy our constraints. However, this will decrease $\min \sum_{a_i=d} x_i - \sum_{b_i=d} r_i x_i$, which will imply that our original solution was not optimal, which is a contradiction. Therefore, there can be no paths ending in a currency other than Yen, which shows that one can end the day with the optimal amount of Yen and no other currency except dollars. □

6.854 Advanced Algorithms

Problem Set 6

John Wang
Collaborators:

Problem 6-a: Argue that at any vertex M of the polytope described above, at least $m - 2n + 1$ of the x_{ij} must be equal to 0.

Solution: First, we notice that there are only $2n$ constraints on the vertices, one for each vertex in V_1 and one for each vertex in V_2, where V_1 and V_2 denote the two subsets of the bipartite matching. However, we will notice that these $2n$ constraints are not linearly independent. This is because summing up all of the constraints for vertices in V_1 will be the sum of all the constraints for vertices in V_2. Formally, we will have $\sum_i \sum_{j \in N(i)} x_{ij} = \sum_j \sum_{i \in N(i)} x_{ij}$.

From this, we see that the two sets of constraints are not linearly independent, so that there are at most $2n - 1$ linearly independent constraints. Since there are m entries in the matrix M, we know that at least $m - 2n + 1$ of the constraints are linearly dependent. Since linear dependence implies that an entry be equal to zero, we know that there are $m - 2n + 1$ of the x_{ij}'s which are equal to 0. □

Problem 6-b: Conclude that at least one row and at least one column of the matrix M contains a single 1 with all other entries 0, if M represents a vertex.

Solution: We notice that there are only $2n - 1$ possible entries in the matrix M which can be non-zero. Note that the matrix is $n \times n$, and we have the constraint that each column sum and each row sum must equal 1. We will first prove the assertion for columns, and the assertion for rows will follow similarly.

Since each column sum must equal 1, we must have at least one non-zero entry per column, where each entry x_{ij} must have $0 < x_{ij} \leq 1$ (since there are no negative numbers). There are now only $n - 1$ entries left to distribute among n columns. There is no way to distribute these $n - 1$ entries so that all n columns receive an additional non-zero entry. This means that one column must have a single non-zero entry. Since we must have $\sum_{i \in N(i)} x_{ij} = 1$, that non-zero entry must be a 1. We have shown, then, that there is at least one column which contains a single 1, with all other entries 0.

The same reasoning can be applied to the rows of matrix M, so we see that at least one column and at least one row contains a single 1 with all other entries 0. □

Problem 6-c: Use induction to conclude the claim (about convex combinations) and deduce the original claim (about internet switches).

Solution: First, we will show that every doubly stochastic matrix is a convex combination of perfect matchings. We will note that $n \times n$ permutation matrices are in one-to-one correspondence with perfect matchings, since each perfect matching picks a single $x_{ij} = 1$ in each row and column and forces all other x_{ij} in that row or column to be 0. This is the definition of a $n \times n$ permutation matrix, so we are left to show that every doubly stochastic matrix M is a convex combination of all $n \times n$ permutation matrices.

We will show that for each doubly stochastic matrix M, we can find a matrix P which contains one non-zero entry from each row and each column of M. We shall create a bipartite graph G where the rows r_i of M are the left set and the columns c_j of M are the right set. There is an edge between row r_i and column c_j if entry x_{ij} is non zero. Now suppose there is no perfect matching in this graph G. This means that there is a set of k rows in the left set which have non-zero entries only in some l columns, where we have $l < k$. However, if we sum the entries in these k rows, we know that each row must have a row sum of 1, so the total sum of these entries is k. Summing the entries over the columns, we know that each column must also sum to 1, so we have a total sum of these entries of s. This implies $k = s$ since we are summing the same things. This is a contradiction of the fact that $s < k$, and hence, there must be a perfect mathing in G.

Now, let P be the matrix which contains one non-zero entry from each row and each column of M. Let P_1 be its corresponding permutation matrix and c_1 be the minimum entry in P. We can now find a new matrix

$M_1 = M - c_1 p_1$ which is non-negative and has equal row and column sums. Since our above analysis for finding a matrix P which contains one non-zero entry from each row and column of M works for whenever M has equal column and row sums (change row and column sums of 1 to some arbitrary u in the proof), we can repeat the procedure on M_1. This will give us a new matrix $M_2 = M - c_1 P_1 - c_2 P_2$. We will eventually terminate this procedure because at each step, we are eliminating an entry and setting that entry to zero. This occurs because we always pick c_1 as the minimum element in P, so that $M_i - c_i P_i$ will set the minimum entry in M_i to zero. Since the new bipartite graph is only built with edges of non-negative entries, that entry will always stay at zero during each step of the algorithm. Finally, since the algorithm eliminates an entry at each step, we will be sure that after $O(n^2)$ iterations, we will have a found $M - c_1 P_1 - \ldots - c_k P_k = 0$.

But this means we have decomposed M into a convex combination of permutation matrices, since $M = c_1 P_1 + \ldots + c_k P_k$. This shows that every doubly stochastic matrix is a convex combination of perfect matchings.

Now, we shall show that so long as the internet switch can deliver matchings at rate λ, then it can deliver the specified traffic. This follows quickly from the fact that a doubly stochastic matrix is a convex combination of perfect matchings. Since at each step, the internet traffic switch can only produce matchings, we only need to show that the number of matchings that M can be decomposed into is less than λ^2 (since we deliver λ matchings per unit of time). However, we know that λ is the maximum rate of input or output, which means that the decomposition algorithm used above will only need λ^2 decompositions. This shows that if the internet switch can deliver matchings at rate λ, then by the decomposition of inputs and outputs, it can deliver the specified amount of traffic. $\square$