CS 170 Homework 9

1 Study Group

List the names and SIDs of the members in your study group. If you have no collaborators, explicitly write "none".

2 Mittens

You are running Toasty Digits, a company that produces mittens. To make sure that your company can meet demands, you are planning out the production for the coming n months. The following information is given to you:

- **Demand:** In month i, the demand will be $d_i \geq 0$ (i.e. you sell exactly d_i pairs of mittens in month i). The total demand for all n months is given by $D = \sum_{i=1}^{n} d_i$.
- **Production:** Your full-time knitting staff can produce at most m pairs of mittens per month. You can hire additional knitters who will produce as many additional mittens as you need, but at a cost of c dollars per pair (whereas you do not pay anything per pair for your full-time staff).
- **Storage:** If, at the end of a month, you have any leftover mittens, you have to store them at a cost. In particular, if you have k pairs of mittens left, you pay $h_k \geq 0$ dollars. You can store at most D pairs of mittens.

Provide a dynamic programming algorithm for computing the minimum total cost required to meet all demand. Your solution should include the following:

(a) A description of your subproblems and how you will use them to compute the final answer.

Hint: Let one of the subproblem parameters be the number of leftover mittens.

(b) A recurrence relation for your subproblems and the relevant base cases.

(c) A justification for your recurrence relation and your base cases.

(d) The order in which to solve the subproblems.

(e) The runtime of solving all subproblems and computing the final answer.

Solution:

(a) We will define the subproblem $f(i, r)$ as the minimum cost to get to the end of month i, having satisfied all demand so far, with r pairs of mittens left over. The final answer will be $f(n, 0)$: the subproblem corresponding to month n with 0 mittens left.

(b) The recurrence is given by

$$f(i, r) = \min_{0 \leq p \leq r + d_i} F(i, r, p)$$

where $F(i, r, p)$ is the cost incurred during month i if we start it with p pairs of mittens and end it with r pairs left:

$$F(i,r,p) = \begin{cases} h_p + f(i-1,p) & \text{if } m \geq r + d_i - p \\ h_p + c(r + d_i - p - m) + f(i-1,p) \end{cases}$$

The base cases are $f(0, \cdot) = \infty$, except for $f(0, 0) = 0$

(c) The base cases follow because at the beginning of month 1, we have zero mittens. This makes $f(0, 0)$ the actual starting point and all other $f(0, \cdot)$ impossible.

The recurrence follows because we want to minimize the cost given the number of mittens we have left over after each month. The cost for each month depends on the demand and the number of mittens we have left over and the number we want to have left over. Furthermore, if we have p mittens left over from the month before, we always pay h_p. Furthermore, we need to consider whatever we have paid so far. Finally, if our full-time staff cannot knit enough mittens to cover demand and the number of mittens we want to have left over, even using the previous stock, we must hire outside staff at cost c per pair.

(d) The order of the subproblems need not depend on the number of mittens left, but we need to solve in terms of increasing months, i.e. $f(i-1, r)$ before $f(i, m)$ for all r, m.

(e) We have $O(nD)$ subproblems and each subproblem takes $O(D)$ time to compute. Hence, the runtime is $O(nD^2)$.

3 Simplex Practice

(a) Consider the following linear program:

$$\begin{aligned} \text{Maximize } & x + y \\ \text{subject to: } & -\frac{1}{2}x + y \leq 3 \\ & x + 2y \leq 12 \\ & y \leq 4 \\ & 3x + y \leq 21 \\ & x, y \geq 0 \end{aligned}$$

Draw the feasible region. Write out the sequence of vertices traversed by the simplex algorithm starting at $(0, 3)$.

(b) Consider the following linear program on three variables:

$$\begin{aligned} \text{Maximize } & f(x, y, z) \\ \text{subject to: } & x + 2y \leq 10 \\ & z \leq 3 \\ & x, y, z \geq 0 \end{aligned}$$

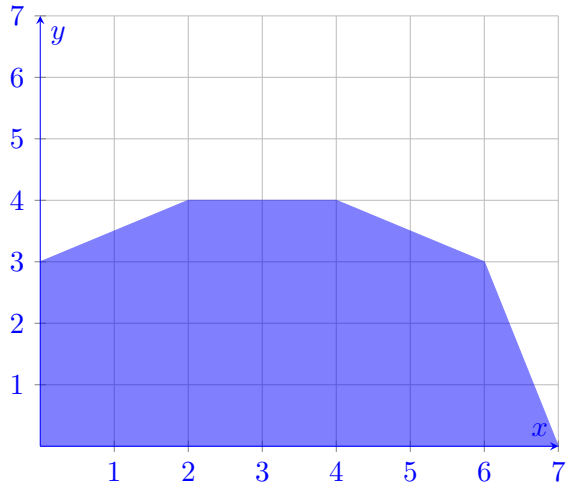
Is it possible for $p = \{(0, 0, 0), (0, 5, 0), (10, 0, 0), (10, 0, 3)\}$ to be a valid path that the simplex algorithm takes for some linear function f? If so, is there a linear function, f, such that p is the *only* possible valid path that the simplex algorithm may take?

(c) Over all linear programs in two variables and n constraints (including $x_1, x_2 \geq 0$), determine the minimum and the maximum number of vertices that the simplex algorithm may need to traverse over including its starting and ending points. You may assume that we start already at some vertex on the feasible region, $\vec{x}$.

(d) In the simplex algorithm, we can choose *any* better vertex upon each iteration. What if we make the following modification: choose the *best* adjacent vertex. How does this change the answer to part (c)?

Solution:

(a)



After starting at $(0, 3)$, we next visit $(2, 4)$ then $(4, 4)$ and finally $(6, 3)$ before our algorithm terminates.

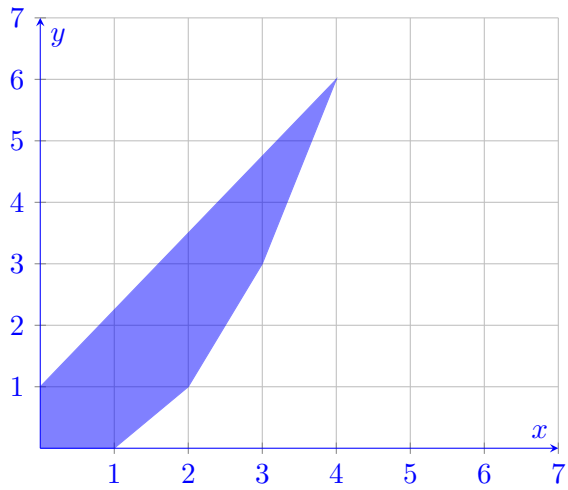
(b) First, it is possible for p to be a valid path! For example, consider $f(x, y, z) = x + y + z$. However, there is no such linear function such that p is the only valid path. Notice that $(0, 0, 0)$ is adjacent to both $(0, 5, 0)$ and $(10, 0, 0)$, but also that $(0, 0, 0)$ is adjacent than $(10, 0, 0)$. So, any time p is a valid path, then $(10, 0, 0)$ is also better than $(0, 0, 0)$, so the path $p' = \{(0, 0, 0), (10, 0, 0), (10, 0, 3)\}$ is also valid.

(c) The minimum number of vertices to iterate on is 1 as we may already be at the maximum. The maximum number of points we may travel over is n. One example uses the constraints in part (a) and an optimization function like $100x + y$. If we start from $(0, 0)$, we may travel to $(0, 3)$, then $(2, 4), (4, 4), (6, 3), (7, 0)$. We cannot traverse a vertex more than once because every iteration, we move to a vertex with strictly larger objective value.

(d) The minimum number of vertices is clearly still 1. As for the maximum, it is now $n - 1$.

We cannot travel over all n vertices. If that were the case, we'd travel to them in a cycle, and end at a vertex that is adjacent to the first vertex. So by our modified algorithm, we would simply travel to the optimal vertex first.

However, it might be the case that $n-1$ vertices are traversed. Intuitively, this is because the worst vertex may be connected to the best vertex, so the simplex algorithm will have to go the other direction (starting at one of the "next-worst" vertices). For example, consider the following feasible region where we start at $(0, 0)$ and the optimization function is x:



In general, the following linear program would work for all n (a generalization of the example):

$$\begin{aligned}
&\text{Maximize } x \\
&\text{subject to: } (y-0) - 1(x-1) \geq 0 \\
&\qquad\qquad (y-1) - 2(x-2) \geq 0 \\
&\qquad\qquad (y-3) - 3(x-3) \geq 0 \\
&\qquad\qquad \vdots \\
&\qquad\qquad \left(y - \binom{n}{2}\right) - n(x-n) \geq 0 \\
&\qquad\qquad (y-1) + \frac{\binom{n}{2} - 1}{n} x \leq 0 \\
&\qquad\qquad x, y \geq 0
\end{aligned}$$

The simplex algorithm starting at $(0,0)$ would travel to $(1,0)$, then $(2,1)$, and so on, before reaching $\left(n, \binom{n}{2}\right)$.

4 Max-Flow Min Cut Basics

For each of the following, state whether the statement is True or False. If true provide a short proof, if false give a counterexample.

(a) If all edge capacities are distinct, the max flow is unique.

(b) If all edge capacities are distinct, the min cut is unique.

(c) If all edge capacities are increased by an additive constant, the min cut remains unchanged.

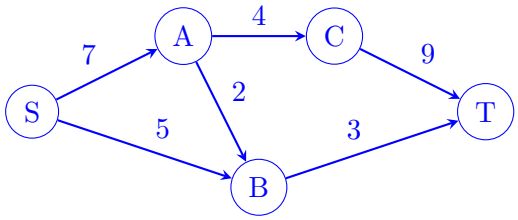
(d) If all edge capacities are multiplied by a positive integer, the min cut remains unchanged.

(e) In any max flow, there is no directed cycle on which every edge carries positive flow.

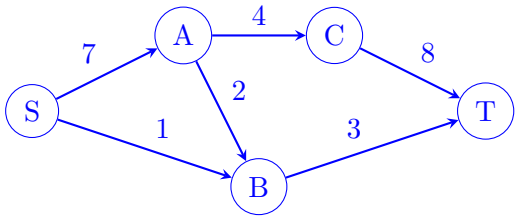
(f) There exists a max flow such that there is no directed cycle on which every edge carries positive flow.

Solution:

(a) False. Consider the following graph:



(b) False. Consider the graph below. The cut SA and the cut SAB are both size 7.

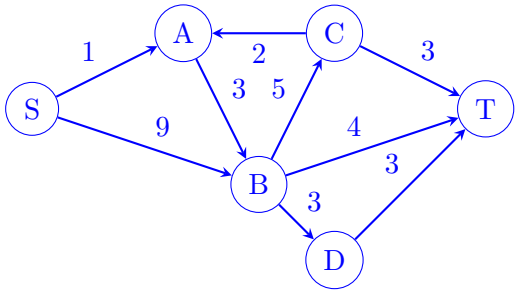


(c) False. Add one to all of the edge capacities of the graph in part (b). The cut SA and the SAB have different values now.

(d) True. Let the value of a cut be $\sum_e c_e$ and the value of the minimum cut be $\sum_{e'} c_{e'}$. The minimum cut must still be the minimum cut after multiplying the edges by a positive constant, due to the distributive property:

$$a\sum_{e'} c_{e'} - a\sum_{e} c_e = a(\sum_{e'} c_{e'} \sum_{e} c_e)$$

(e) False. Consider the graph below:



(f) True. See any graphs other than (e). Consider the graph in part (e) without the edge SA.

5 Flow vs LP

You play a middleman in a market of m suppliers and n purchasers. The i-th supplier can supply up to $s[i]$ products, and the j-th purchaser would like to buy up to $b[j]$ products.

However, due to legislation, supplier i can only sell to a purchaser j if they are situated at most 1000 miles apart. Assume that you're given a list L of all the pairs (i, j) such that supplier i is within 1000 miles of purchaser j. Given m, n, $s[1..m]$, $b[1..n]$, and L as input, your job is to compute the maximum number of products that can be sold. The runtime of your algorithm must be polynomial in m and n.

Show how to solve this problem using a network flow algorithm as a subroutine. Describe the graph and explain why the output from running the network flow algorithm on your graph gives a valid solution to this problem.

Solution:

(a) *Algorithm:* We create a bipartite graph with $m + n + 2$ nodes. Label two of the nodes as a "source" and a "sink." Label m nodes as suppliers, and n nodes as purchasers. Now, we will create the following edges:

- Create an edge from the source to supplier i with capacity $s[i]$.
- For each pair (i, j) in L, create an edge from supplier i to purchaser j with infinite capacity.
- Create an edge from purchaser j to the sink with capacity $b[j]$. We then plug this graph into our network flow solver, and take the size of the max flow as the number of dollars we can make.

Proof of correctness: We claim that the value of the max flow is precisely the maximum amount transactions we can make. To show this, we can show that a strategy of selling products corresponds exactly to a flow in this graph and vice versa.

For any flow, let $x_{i,j}$ be the amount of flow that goes from the node of supplier i to the node of purchaser j. Then, we claim a product selling strategy will sell exactly $x_{i,j}$ products from supplier i to purchaser j. This is a feasible strategy, since the flow going out of the node of supplier i is already bounded by $s[i]$ by the capacity of the edge from the source to that node, and similarly for the purchasers. Similarly, for the other direction, one can observe that any feasible selling strategy leads to a feasible flow of the same value.

(b) We define a variable $x_{i,j}$, denoting the amount of products we take from supplier i and

sell to purchaser j. Then, we have the following linear program

$$\begin{aligned}
\max \sum_{i=1}^{m}\sum_{j=1}^{n} x_{i,j} \\
\sum_{i=1}^{m} x_{i,j'} \leq b[j'], \text{ for all } j' \in [1,n] \\
\sum_{j=1}^{n} x_{i',j} \leq s[i'], \text{ for all } i' \in [1,m] \\
x_{i,j} = 0, \text{ for all } (i,j) \notin L \\
x_{i,j} \geq 0, \text{ for all } (i,j)
\end{aligned}$$

The linear program has mn variables and $O(mn)$ linear inequalities, so it can be solved in time polynomial in m and n.

(c) Network flow is better. Linear programming is not guaranteed to find an integer solution (not even if one exists), so the approach in part (b) might yield a solution that would involve selling fractional products. In contrast, since all the edge capacities in our graph in part (a) are integers, the Ford-Fulkerson algorithm for max flow will find an integer solution. Thus, max flow is the better choice, because there are algorithms for that formulation that will let us find an integer solution.

6 A Cohort of Secret Agents

A cohort of k secret agents residing in a certain country needs escape routes in case of an emergency. They will be travelling using the railway system which we can think of as a directed graph $G = (V, E)$ with V being the cities and E being the railways. Each secret agent i has a starting point $s_i \in V$, and all s_i's are distinct. Every secret agent needs to reach the consulate of a friendly nation; these consulates are in a known set of cities $T \subseteq V$. In order to move undetected, the secret agents agree that at most c of them should ever pass through any one city. Our goal is to find a set of paths, one for each of the secret agents (or detect that the requirements cannot be met).

Model this problem as a flow network. Specify the vertices, edges, and capacities; show that a maximum flow in your network can be transformed into an optimal solution for the original problem. You do not need to explain how to solve the max-flow instance itself.

Solution: We can think of each secret agent i as a unit of flow that we want to move from s_i to any vertex $\in T$. To do so, we can model the graph as a flow network by setting the capacity of each edge to ∞, adding a new vertex t and adding an edge (t', t) for each $t' \in T$. We can add a source s and edges of capacity 1 from s to s_i. By doing so, we restrict the maximum flow to be k.

Lastly, we need to ensure that no more than c secret agents are in a city. We want to add vertex capacities of c to each vertex. To implement it, we do the following: for each vertex v that we want add constraint to, we create 2 vertices v_{in} and v_{out}. v_{in} has all the incoming edges of v and v_{out} has all the outcoming edges. We also put a directed edge from v_{in} to v_{out} with edge capacity constraint c.

If the max flow is indeed k, then as every capacity is an integer, the Ford-Fulkerson algorithm for computing max flow will output an integral flow (one with all flows being integers). Therefore, we can incrementally starting at each secret agent i follow a path in the flow from s_i to any vertex in T. We iterate through all of secret agents to reach a solution.

7 Office Hour Scheduling

You're the new CS 170 head TA and need to assign TAs to host office hours. Unfortunately for you, the department has bizarre rules about when TAs can hold office hours:

- There are N office hours slots in total. You can assume that all N slots are consecutive.
- There are T TAs in total. The i^{th} TA is only available to hold office hours during the contiguous block from slot a_i to slot b_i. The numbers $\{a_i, b_i\}$ for $i \in \{1 \dots T\}$ are given as input.
- Each office hours slot must have exactly k TAs (no more, and no less).
- Not every TA needs to hold office hours.
- If a TA holds office hours, department rules require them to hold office hours for their entire availability, e.g. the i^{th} TA either holds office hours for every slot from a_i to b_i, or does not hold office hours at all.

Your goal is to determine which TAs to assign office hours so that all the above constraints are met.

Devise an efficient algorithm that solves the problem by constructing a directed graph G with a source s and sink t, and computing the maximum flow from s to t.

(a) Describe the nodes of the graph G. How many nodes does G have?

(b) Describe the edges of the graph G (draw a picture if needed). How many edges does G have?

(c) Given a maximum flow in the graph G, how can you determine which TAs to assign to hold office hours?

Solution:

(a) **Solution 1:** We create one vertex for each OH slot from $n = 1 \dots N$. We also create a start node s and an end node t, which we will respectively index as 0 and $N + 1$ for simplicity of the arguments below.

There are $N + 2$ nodes in this graph.

Solution 2: We create 2 vertices x_i, y_i for each TA from $i = 1 \dots T$, plus s and t.

There are $2T + 2$ nodes in this graph.

(b) **Solution 1:** Create an edge $u \to v$ for $u, v \in \{0, \dots, N\}$ if there is some TA who is available from slot $u + 1$ to slot v, and set its capacity to the number of TAs with this availability range. Note that we take vertex s to have index 0 for the purpose of these calculations. Additionally, add an edge from vertex N to the end node t with capacity k.

There are $T + 1$ edges in this graph.

Solution 2: For a given pair of TAs i, j, create an edge $y_i \to x_j$ with unit capacity if $b_i + 1 = a_j$. Create an edge $x_i \to y_i$ with unit capacity for all TAs $i \in \{1, \ldots, T\}$. Connect s to all vertices x_i with $a_i = 0$, and connect all y_i s.t. $b_i = N$ to t.

There are $O(T^2)$ edges in this graph.

(c) **Solution 1:** first, check if the value of the max flow is k; if it's not, then it's impossible to assign TAs that satisfies all the constraints.

Otherwise, we can look at the residual graph and the flow (i.e. residual) along each edge (u, v) will correspond to the number of TAs we want to assign to the OH slots $u + 1$ to v. Based on this, we can assign TAs to office hour slots accordingly.

Solution 2: first, check if the value of the max flow is at least k; if it's not, then it's impossible to assign TAs that satisfies all the constraints.

Otherwise, let f^* be the value of the max-flow. We can decompose our maximum flow into a set of exactly f^* disjoint $s - t$ paths, each with flow exactly 1. We can choose k of these paths, and assign each vertex (i.e. TA) along each of the paths to hold OH.

Note: Solution 2 is preferable to the other solution because N is only given as a parameter (whereas we are given a list of T TAs), so solution 1 is actually pseudo-polynomial time. Both will be given full points, however.