COMS20010 — Problem sheet 7

You don't have to finish the problem sheets before the class — focus on understanding the material the problem sheet is based on. If working on the problem sheet is the best way of doing that, for you, then that's what you should do, but don't be afraid to spend the time going back over the quiz and videos instead. (Then come back to the sheet when you need to revise for the exam!) I'll make solutions available shortly after the problem class. As with the Blackboard quizzes, question difficulty is denoted as follows:

- ⋆ You'll need to understand facts from the lecture notes.
- ⋆⋆ You'll need to understand and apply facts and methods from the lecture notes in unfamiliar situations.
- ⋆⋆⋆ You'll need to understand and apply facts and methods from the lecture notes and also move a bit beyond them, e.g. by seeing how to modify an algorithm.
- ⋆⋆⋆⋆ You'll need to understand and apply facts and methods from the lecture notes in a way that requires significant creativity. You should expect to spend at least 5–10 minutes thinking about the question before you see how to answer it, and maybe far longer. Only 20% of marks in the exam will be from questions set at this level.
- ⋆⋆⋆⋆⋆ These questions are harder than anything that will appear on the exam, and are intended for the strongest students to test themselves. It's worth trying them if you're interested in doing an algorithms-based project next year — whether you manage them or not, if you enjoy thinking about them then it would be a good fit.

If you think there is an error in the sheet, please post to the unit Team — if you're right then it's much better for everyone to know about the problem, and if you're wrong then there are probably other people confused about the same thing.

This problem sheet covers week 7, focusing on linear programming and network flows.

1. (a) [⋆⋆] Convert the following linear programming problem into standard form.

$$\begin{aligned} 4x - 3y + z &\to \min \text{ subject to} \\ x + y &\leq 4, \\ 3x - 2y - z &\geq 2, \\ x,\, y &\geq 0. \end{aligned}$$

Solution: Using the method described in lectures, we obtain

$$-4x + 3y - z_1 + z_2 \to \max \text{ subject to}$$

$$\begin{pmatrix} 1 & 1 & 0 & 0 \\ -3 & 2 & 1 & -1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z_1 \\ z_2 \end{pmatrix} \leq \begin{pmatrix} 4 \\ -2 \end{pmatrix},$$

$$x,\, y,\, z_1,\, z_2 \geq 0.$$

(b) [$\star\star\star$] Some linear programming algorithms prefer to take their input in another form, known as *slack form*. In slack form, we are given a linear objective function $f\colon \mathbb{R}^n \to \mathbb{R}$, an $m \times n$ matrix A, and an m-dimensional vector $\vec{b} \in \mathbb{R}^m$. The desired output is a vector $\vec{x} \in \mathbb{R}^n$ maximising $f(\vec{x})$ subject to $A\vec{x} = \vec{b}$ and $\vec{x} \geq 0$; thus slack form is the same as standard form, except that the upper bound constraint $A\vec{x} \leq \vec{b}$ is replaced by the equality constraint $A\vec{x} = \vec{b}$.

Put the linear programming problem from part (a) into slack form, and explain how to apply your method to convert any problem in standard form into slack form. (**Hint:** You may find it useful to add new variables.)

Solution: In general, to turn a problem in standard form into slack form, we add one new *slack variable* to each constraint — say the i'th constraint gets variable s_i. We then change the constraint from $\sum_j A_{i,j}x_j \leq b_i$ to $\sum_j A_{i,j}x_j + s_i = b_i$, and add non-negativity constraints $s_i \geq 0$. Then for each value of $\vec{x}$ with $\sum_j A_{i,j}x_j \leq b_i$, there is a unique value of s_i with $\sum_j A_{i,j}x_j + s_i = b_i$; conversely, since $s_i \geq 0$, if $\sum_j A_{i,j}x_j + s_i = b_i$ then we also have $\sum_j A_{i,j}x_j \leq b_i$. Thus the two problems are equivalent. For the specific problem above, the slack form is

$$-4x + 3y - z_1 + z_2 \to \max \text{ subject to}$$

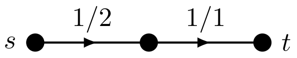
$$\begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 \\ -3 & 2 & 1 & -1 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z_1 \\ z_2 \\ s_1 \\ s_2 \end{pmatrix} = \begin{pmatrix} 4 \\ -2 \end{pmatrix},$$

$$x,\, y,\, z_1,\, z_2,\, s_1,\, s_2 \geq 0.$$

2. Are the following statements true or false? For each one, give either a short explanation (if it's true) or a counterexample (if it's false).

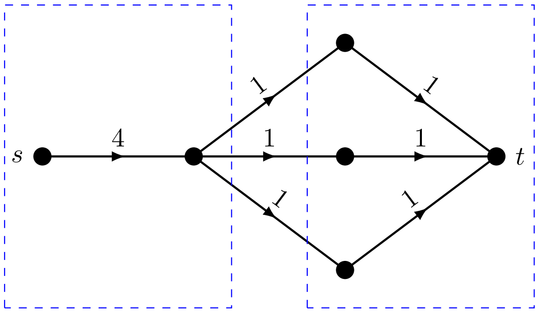
(a) [$\star\star$] If f is a maximum flow in a flow network (G, c, s, t), then $f(e) = c(e)$ for all edges e incident to the source s.

Solution: This is false — here's a counterexample, shown with its maximum flow:

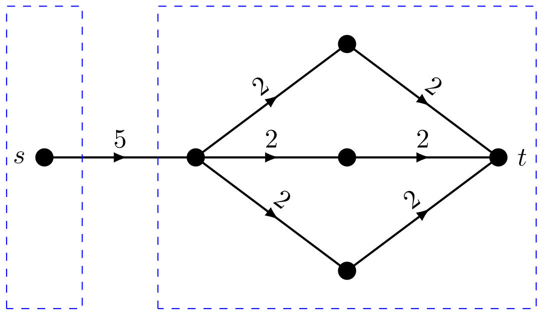


(b) [$\star\star\star$] If (A, B) is a minimum cut in a flow network (G, c, s, t), and we add 1 to the capacity of every edge, then (A, B) is still a minimum cut.

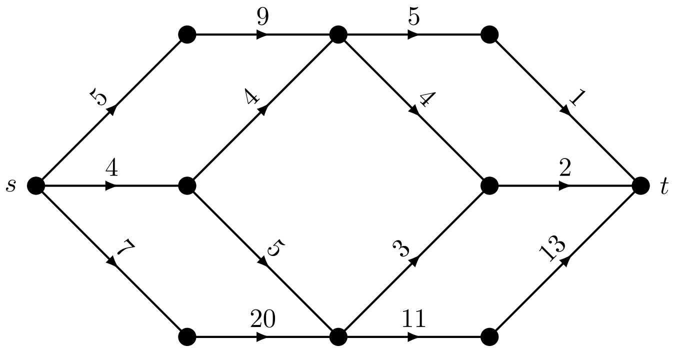
Solution: This is false — here's a counterexample, shown a minimum cut of the original network:



But in the network with increased capacities, this cut has value 6 and there is a cut with value 5:

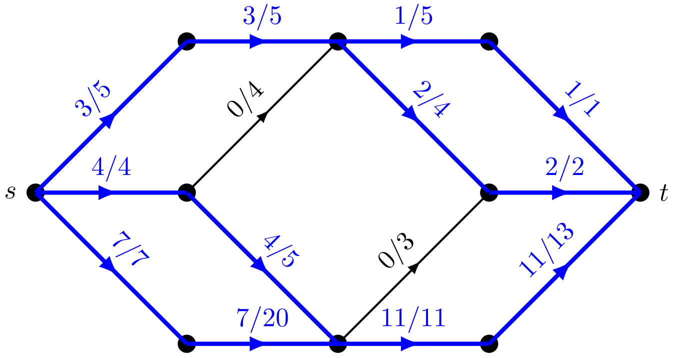


3. Consider the following flow network with capacities indicated:



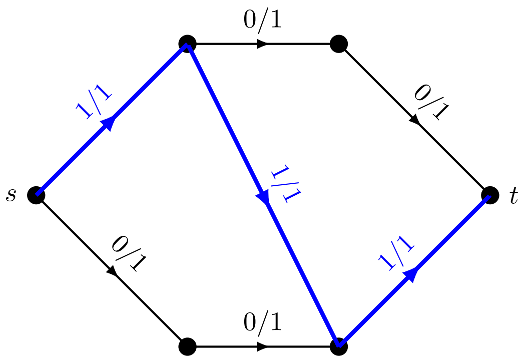
(a) [$\star\star$] Run the Ford-Fulkerson algorithm on this flow network to find a maximum flow, showing how the residual network evolves as the current flow changes.

Solution:



This is a maximum flow with $v(f) = 14$.

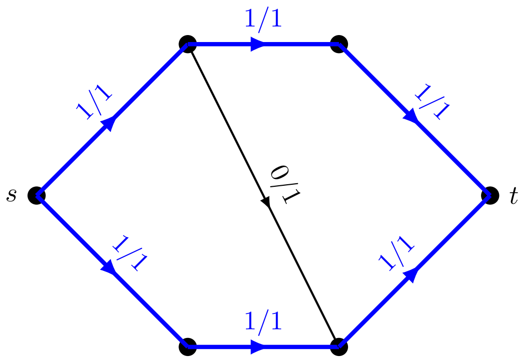
(b) [$\star\star$] Now consider the following flow network with a flow.



Continue running the Ford-Fulkerson algorithm on this flow network to find a maximum flow, showing how the residual network evolves as the current flow changes.

Hint: How are reverse edges useful here?

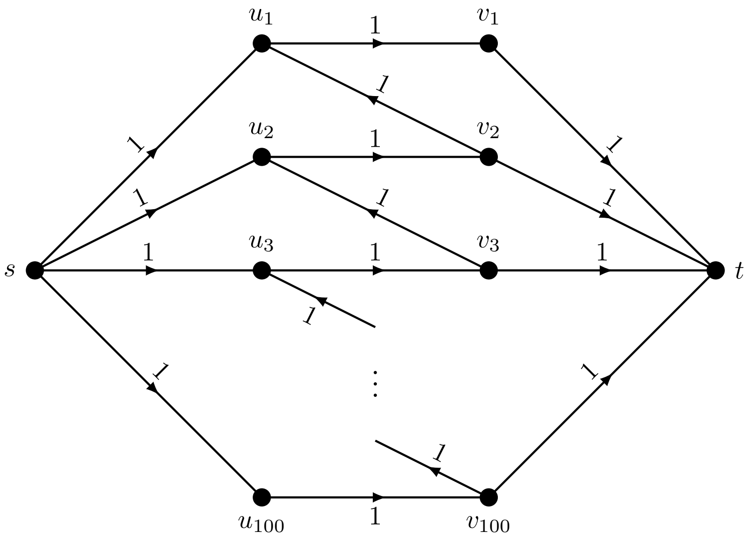
Solution: A maximum flow for this network looks like



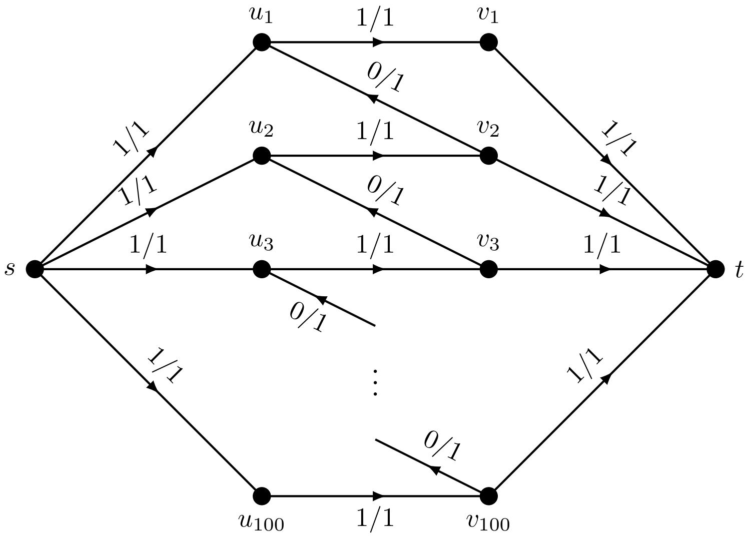
with $v(f) = 2$.

4. [⋆⋆⋆⋆] John thinks backward edges sound unnecessarily complicated, and so he decides to implement the Ford-Fulkerson algorithm using only forward edges. (In other words, his algorithm only chooses augmenting paths that increase flow along every edge, and terminates when no such paths exist.) He reasons that while it won't always give a maximum flow, it will at least give a decent approximation. Show that John is wrong by giving an example on which his algorithm may output a flow whose value is at most one hundredth of that of a maximum flow. You may make whatever assumptions you like about how John's algorithm chooses its augmenting paths. (**Hint**: Your example may have too many vertices to draw in full.)

Solution: The following example works, where s is the source and t is the sink.



John's algorithm may take its first augmenting path to be $su_{100}v_{100}u_{99}v_{99}u_{98}v_{98}\dots u_2v_2u_1v_1t$. John's algorithm will then push flow down that path, resulting in a value-1 flow in which there are no more augmenting paths containing only forward edges. John's algorithm will then output that flow. However, the flow pictured below has value 100:



so the output's value is at most one hundredth that of a maximum flow, as required.

5. (a) [★★★] Prove that the approximation algorithm for vertex cover given in lectures works, outputting a vertex cover with weight at most twice the minimum possible.

Solution: This is the same as the solution to part b), but with the weights set to 1.

(b) [★★★] In the *weighted* vertex cover problem, we are given a graph $G = (V, E)$ and a weight function $w: V \to \mathbb{N}$, and we must output a vertex cover with minimum total weight. Adapt the unweighted algorithm given in lectures to solve this problem approximately, and prove it works.

Solution: We form the linear program relaxation as follows:

$$\begin{aligned}\sum_{v\in V} w(v)x_v &\to \min \text{ subject to}\\ x_u + x_v &\geq 1 \text{ for all } \{u,v\}\in E,\\ x_v &\leq 1 \text{ for all } v\in V,\\ x_v &\geq 0 \text{ for all } v\in V.\end{aligned}$$

Exactly as in the unweighted problem, integer solutions correspond to vertex covers (taking v to be in the cover if and only if $x_v = 1$), and an optimal integer solution would correspond to a minimum-weight vertex cover. Again as in the unweighted case, we solve this linear program with an algorithm of our choice, round each x_v to the nearest integer, and output the resulting vertex set. Let X be the set we output, let $\{x'_v : v\in V\}$ be the result of rounding our linear program solution, let $w(X) = \sum_{v\in X} w(v)$, and let w^* be the minimum possible weight of a vertex cover. We must show that X is a vertex cover with $w^* \leq w(X) \leq 2w^*$.

***Y* is a vertex cover:** For every edge $\{u,v\}\in E$, we have $x_u + x_v \geq 1$, so at least one of x_u or y_u must be at least $1/2$. Suppose (without loss of generality) it is x_u. Then $x'_u = 1$, so $u \in X$. We have shown that X contains at least one endpoint of every edge in G, as required. ✓

$w(X) \geq w^*$: This follows since Y is a vertex cover and every vertex cover has weight at least w^*. ✓

$w(X) \leq 2w^*$: Recall that $w(X) = \sum_{v\in X} w(v)$. Since $v\in X$ if and only if $x_v \geq 1/2$, it follows that

$$w(X) = \sum_{\substack{v\in V\\ x_v\geq 1/2}} w(v) = \sum_{\substack{v\in V\\ x_v\geq 1/2}} w(v)x_v \cdot \frac{1}{x_v} \leq 2 \sum_{\substack{v\in V\\ x_v\geq 1/2}} w(v)x_v \leq 2\sum_{v\in V} w(v)x_v.$$

Since any vertex cover is an (integer) solution of the linear program, and x_v is the optimal (real) solution, it follows that $w(X) \leq 2w^*$. ✓

This is a significant benefit of LP relaxation as a technique — if you can use it to approximately solve the unweighted version of a problem, you can often solve the weighted version with essentially the same algorithm.

6. Consider the folowing linear programming problem:

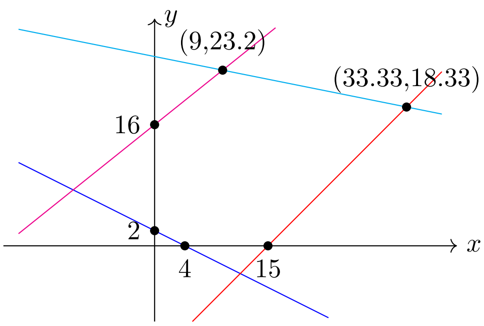
$$\begin{aligned}7x + 8y &\to \max \text{ subject to}\\ -0.5x + 2 &\leq y\\ x - 15 &\leq y\\ -0.2x + 25 &\geq y\\ 0.8x + 16 &\geq y\\ x, y &\geq 0\end{aligned}$$

(a) [⋆⋆] Convert this problem into standard form.

Solution:

$$7x + 8y \to \max \text{ subject to}$$
$$\begin{bmatrix} -0.5 & -1 \\ 1 & -1 \\ 0.2 & 1 \\ 0.8 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \leq \begin{bmatrix} -2 \\ 15 \\ 25 \\ 16 \end{bmatrix}$$
$$x, y \geq 0$$

(b) $[\star\star]$ Plot these constraints. What do they look like geometrically?



Solution:

(c) $[\star\star]$ Evaluate the objective function at each vertex to determine the optimal values of x and y for this problem.

Solution:

$$\begin{aligned} (4, 0) &\to 7x + 8y = 28 \\ (15, 0) &\to 7x + 8y = 105 \\ (33.33, 18.33) &\to 7x + 8y = 380 \\ (9, 23.2) &\to 7x + 8y = 248.6 \\ (0, 16) &\to 7x + 8y = 128 \\ (0, 2) &\to 7x + 8y = 16 \end{aligned}$$

It follows that the optimum is at $(33.33, 18.33)$.

7. (a) $[\star\star]$ Consider the following linear programming problem:

$$\begin{aligned} 3(x_1 + x_2 + x_3 + x_4 + x_5 + x_6) &\to \max \text{ subject to} \\ 5x_1 + 4x_2 + 2x_3 + x_4 &\leq 10, \\ x_3 + 2x_4 + 4x_5 + 5x_6 &\leq 25, \\ x_1, \ldots, x_6 &\geq 0. \end{aligned}$$

By subtracting the left-hand sides of both constraints from the objective function, prove that the optimal solution has value at most 35.

Solution: If $x_1, \ldots, x_6$ satisfy the given constraints, we have

$$\begin{aligned} 3\sum_{i=1}^{6} x_i &\leq 3\sum_{i=1}^{6} x_i - (5x_1 + 4x_2 + 2x_3 + x_4) + 10 - (x_3 + 2x_4 + 4x_5 + 5x_6) + 25 \\ &= -2x_1 - x_2 - x_5 - 2x_6 + 35 \leq 35. \end{aligned}$$

(b) [⋆⋆⋆] Instead of subtracting the left-hand sides of both constraints once each, we could have subtracted a constant multiple a_1 of $5x_1+4x_2+2x_3+x_4$ and a constant multiple a_2 of $x_3+2x_4+4x_5+5x_6$ and then kept the rest of the argument the same, yielding a different and potentially better upper bound. Formulate the problem of trying to find the best possible upper bound with this method as a two-variable linear program in a_1 and a_2.

Solution: For any choice of $a_1, a_2 \geq 0$, we have

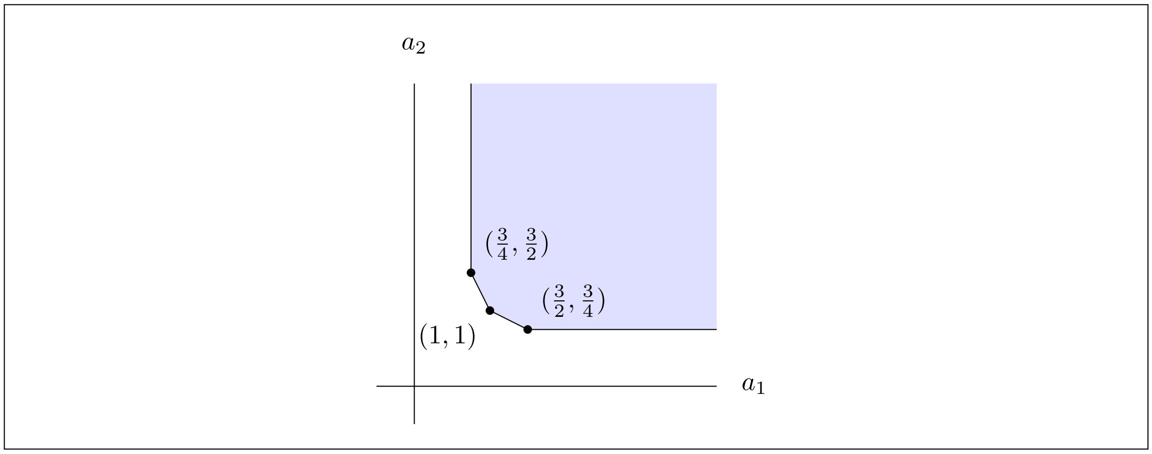
$$\begin{aligned} 3\sum_{i=1}^{6} x_i &\leq 3\sum_{i=1}^{6} x_i - a_1(5x_1 + 4x_2 + 2x_3 + x_4 - 10) - a_2(x_3 + 2x_4 + 4x_5 + 5x_6 - 25) \\ &= (3 - 5a_1)x_1 + (3 - 4a_1)x_2 + (3 - 2a_1 - a_2)x_3 + (3 - a_1 - 2a_2)x_4 \\ &\qquad + (3 - 4a_2)x_5 + (3 - 5a_2)x_6 + (10a_1 + 25a_2). \end{aligned}$$

Since $x_1, \ldots, x_6 \geq 0$, we can conclude $3\sum_{i=1}^{6} x_i \leq 10a_1+25a_2$ **if** each variable x_i has a negative coefficient. Thus to find the best possible upper bound, we must solve the linear program

$$\begin{aligned} 10a_1 + 25a_2 &\to \min \text{ subject to} \\ 3 - 5a_1 &\leq 0, \\ 3 - 4a_1 &\leq 0, \\ 3 - 2a_1 - a_2 &\leq 0, \\ 3 - a_1 - 2a_2 &\leq 0, \\ 3 - 4a_2 &\leq 0, \\ 3 - 5a_2 &\leq 0, \\ a_1, a_2 &\geq 0. \end{aligned}$$

(c) [⋆⋆] Solve this new linear program (e.g. using the fact that the solution must be at a corner of the feasible polytope) to prove that the original problem's optimal solution has value at most 33.75.

Solution: Note that all constraints except the third and fourth are satisfied if and only if $a_1, a_2 \geq 3/4$. The third constraint is equivalent to $a_2 \geq 3 - 2a_1$, and the fourth constraint is equivalent to $a_2 \geq (3 - a_1)/2$. The feasible polytope therefore looks like this:



Plugging in $(\frac{3}{4}, \frac{3}{2})$, $(1,1)$ and $(\frac{3}{2}, \frac{3}{4})$ into our objective function $10a_1 + 25a_2$, we see that it is minimised when $a_1 = \frac{3}{2}$ and $a_2 = \frac{3}{4}$, giving a value of 33.75.

(d) [⋆⋆⋆⋆] Generalise this method to give an upper bound for an arbitrary linear programming problem in standard form. (In fact, this "upper bound" will always be *equal* to the optimal value — the new linear programming problem formed is called the "dual" of the original "primal" problem. This is a key ingredient in almost every algorithm for solving linear programming problems.)

Solution: Consider an arbitrary linear programming problem in standard form:

$$\begin{aligned} \vec{c} \cdot \vec{x} &\to \max \text{ subject to} \\ A\vec{x} &\leq \vec{b}, \\ \vec{x} &\geq \vec{0}. \end{aligned}$$

If A is an $m \times n$ matrix, say, then for all $a_1, \ldots, a_m \geq 0$, we have

$$\begin{aligned} \vec{c} \cdot \vec{x} &\leq \vec{c} \cdot \vec{x} - \sum_{i=1}^{m} a_i (A\vec{x})_i + \sum_{i=1}^{m} a_i b_i \\ &= \sum_{j=1}^{n} \Big(c_j - \sum_{i=1}^{m} a_i A_{i,j}\Big) x_j + \vec{a} \cdot \vec{b} \\ &= \sum_{j=1}^{n} (c_j - (A^T \vec{a})_j) x_j + \vec{a} \cdot \vec{b} \\ &\leq \vec{a} \cdot \vec{b} \text{ if } A^T \vec{a} \geq \vec{c}. \end{aligned}$$

Thus to find the optimum upper bound, we must solve the linear programming problem in $\vec{a}$:

$$\begin{aligned} \vec{b} \cdot \vec{a} &\to \min \text{ subject to} \\ \vec{A}^T \vec{a} &\geq \vec{c}, \\ \vec{a} &\geq \vec{0}. \end{aligned}$$

8. (a) [⋆⋆⋆] You are working in the upper echelons of MI6 during the Cold War. The organisation's spies in Russia communicate via dead drops, parcels left in predetermined locations on a predetermined schedule. For security reasons, each pair of nearby spies has their own dedicated dead drop which no-one knows about but them. SPECTRE is trying to find out where these dead drops are, so they

can intercept the parcels. M is concerned that if SPECTRE finds enough dead drops, they may be able to completely disconnect her spy network, so that at least one pair of spies is completely unable to get in contact with each other even via intermediaries. Give a polynomial-time algorithm which takes as input a list of pairs of spies with dead drops between them and outputs the minimum k such that SPECTRE can disconnect the spy network by compromising k dead drops. (**Hint:** Try reducing the problem to finding minimum cuts in suitably-constructed flow networks.)

Solution: The algorithm is as follows. First construct a graph G whose vertex set is the set of all spies, where spies a and b are joined by an edge if they can communicate via dead drop. Let $c\colon E(G) \to \mathbb{N}$ be the function mapping every edge in G to 1. For each pair s, t of distinct spies, construct the graph $G_{s,t}$ formed from G by directing every edge between s and $N(s)$ towards $N(s)$, and directing every edge between t and $N(t)$ towards t. Then use the Edmonds-Karp algorithm to find the value $v_{s,t}$ of a maximum flow in the network $(G_{s,t}, c, s, t)$. Finally, calculate and output $\min\{v_{s,t}\colon s, t \in V, s \neq t\}$.

It is easy to see that the algorithm runs in polynomial time; it remains to prove that it outputs the correct answer, i.e. the smallest possible size of a set $F \subseteq E(G)$ such that $G - F$ is disconnected with s and t in different components. We call such a set an *st-edge separator*, and claim that the size of a smallest possible edge separator $F_{\min}$ is equal to the capacity of a minimum cut $(X_{\min}, Y_{\min})$ in $(G_{s,t}, c, s, t)$.

$|F_{\min}| \geq c^+(X_{\min})$: Let X be the set of vertices reachable from s in $G - F$. We have $s \in X$ by construction, and since F is a separator we must have $t \notin X$, so $(X, V \setminus X)$ is a cut. Every edge from X to $V \setminus X$ must be included in F, so we have $c^+(X_{\min}) \leq c^+(X) \leq |F_{\min}|$. ✓

$|F_{\min}| \geq c^+(X_{\min})$: Let F be the set of edges between $X_{\min}$ and $Y_{\min}$; thus $|F| = c^+(X_{\min})$. Then since $s \in X_{\min}$ and $t \in Y_{\min}$, removing F from G must disconnect s from t. Thus $|F_{\min}| \leq |F| = c^+(X_{\min})$. ✓

Thus the minimum size of an *st*-edge separator is precisely the capacity of a minimum cut in $(G_{s,t}, c, s, t)$, which by the max-flow min-cut theorem is precisely $v_{s,t}$. Our algorithm is therefore correct.

(b) [⋆⋆⋆⋆] James Bundt suggests another measure of the spy network's robustness: the minimum C such that every pair of spies is joined by C "paths" of dead drops, with no dead drop appearing in more than one path. Show that these two measures coincide, in the sense that $C = k$ in every possible spy network. (**Hint:** Try greedily building these paths from the maximum flows of part (a).)

Solution: Let $C_{s,t}$ be the maximum size of a collection of edge-disjoint paths between s and t in G; then we have $C = \min\{C_{s,t}\colon s, t \in V(G), s \neq t\}$. We claim that $C_{s,t}$ is equal to $v_{s,t}$, i.e. the value of a maximum flow from s to t in the network $(G_{s,t}, c, s, t)$.

Let f be a maximum flow in this network; as proved in lectures, we may assume that it is integer-valued, so that $f(e) \in \{0, 1\}$ for all $e \in E(G_{s,t})$. Let X be the set of all vertices reachable from s along edges with flow 1. Then all edges from S to $V(G) \setminus S$ have flow 0, so if $t \notin S$ then $(S, V(G) \setminus S)$ is a cut with $f^+(S) - f^-(S) \leq f^+(S) = 0$; since $f^+(S) - f^-(S) = v(f) > 0$ (as shown in lectures), this cannot happen. Thus there is a path P_1 from s to t using only edges with flow 1. Reduce flow along those edges by 1, reducing $v(f)$ by 1, and repeat the process $v(f)$ times to obtain $v(f)$ paths from s to t. These paths are clearly edge-disjoint, and f remains a flow throughout the process since we reduce both $f^+(u)$ and $f^-(u)$ by the same amount for all $u \in V(G) \setminus \{s, t\}$. Thus $C_{s,t} = v_{s,t}$ as claimed.

It now follows from the previous part of the question that

$$C = \min\{C_{s,t}\colon s, t \in V(G), s \neq t\} = \min\{v_{s,t}\colon s, t \in V(G)\} = k,$$

as required.