P1: 10 P3: 10
P2: 4 P4: 10
P5: 10
44

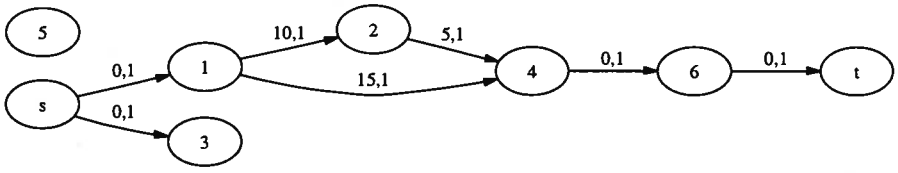
6.854
Advanced Algorithms
2004/10/13

Dan Ports
drkp@mit.edu
Collaborators: {sarahl,pramook}

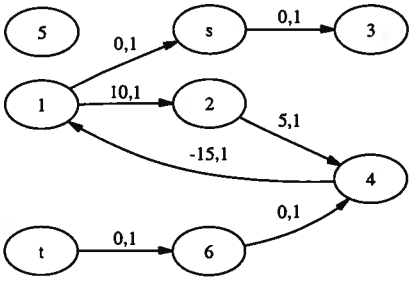
# Problem Set 5

**Problem 1:**

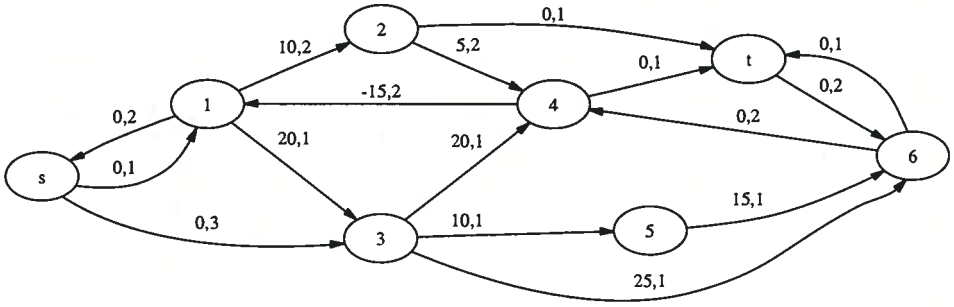
The largest capacity is 15, so four scaling steps will be required. We begin by applying only the highest order bit of the capacity to the graph:



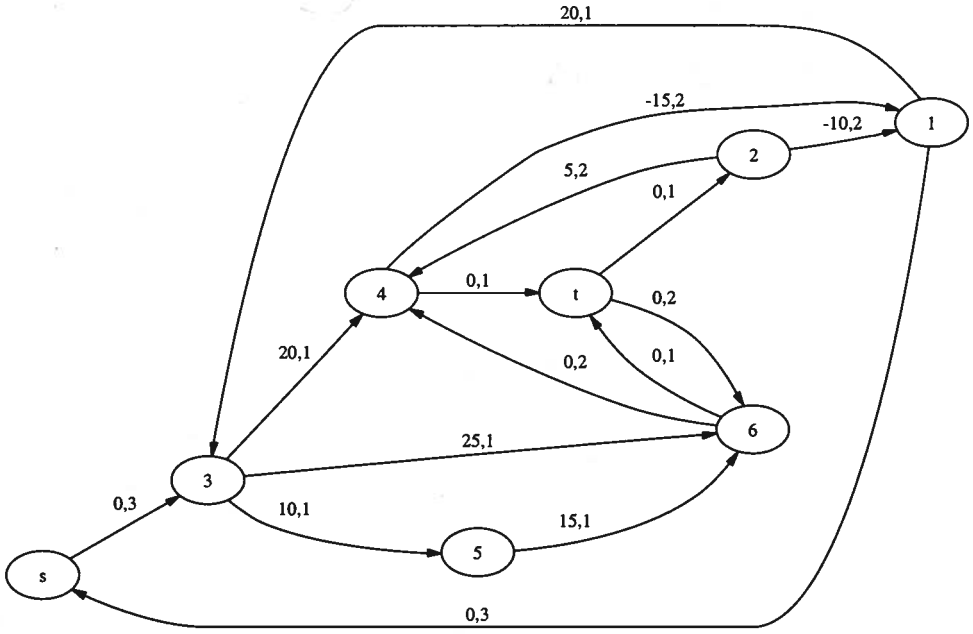

The shortest path is $s \to 1 \to 4 \to 6 \to t$ (or $s \to 1 \to 2 \to 4 \to 6 \to t$, which has the same cost, but we arbitrarily choose the former). We augment along this path and compute the residual network:



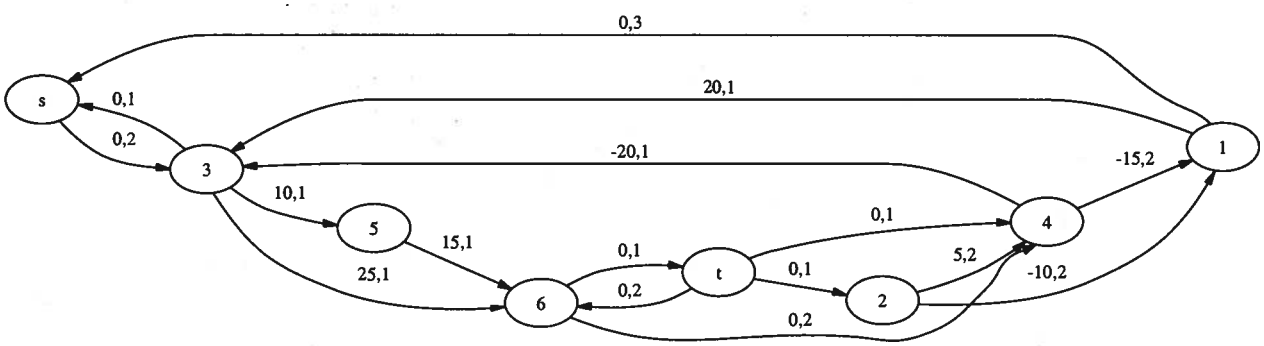

There are no further augmenting paths in the residual network, so we shift in the next bit:



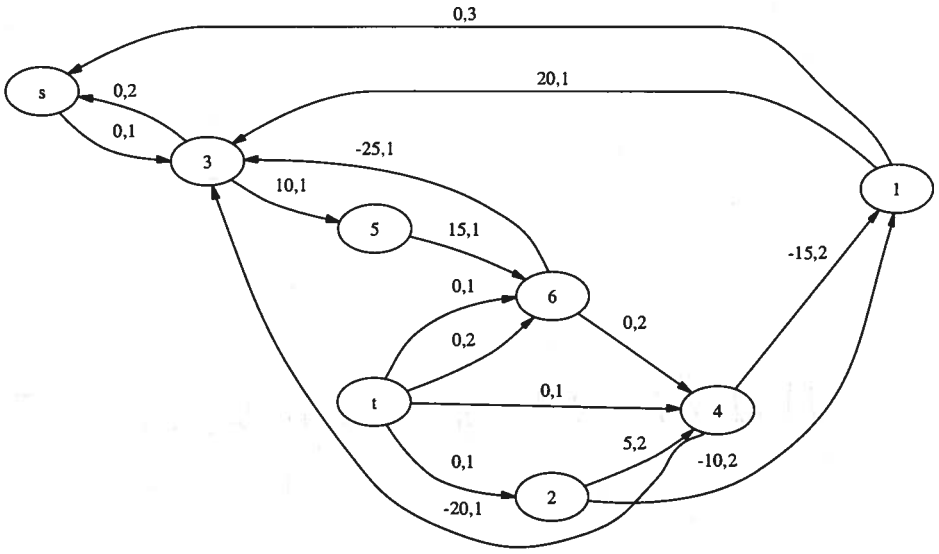

The shortest path is $s \to 1 \to 2 \to t$, with capacity 1 and cost 10. We augment along this path:



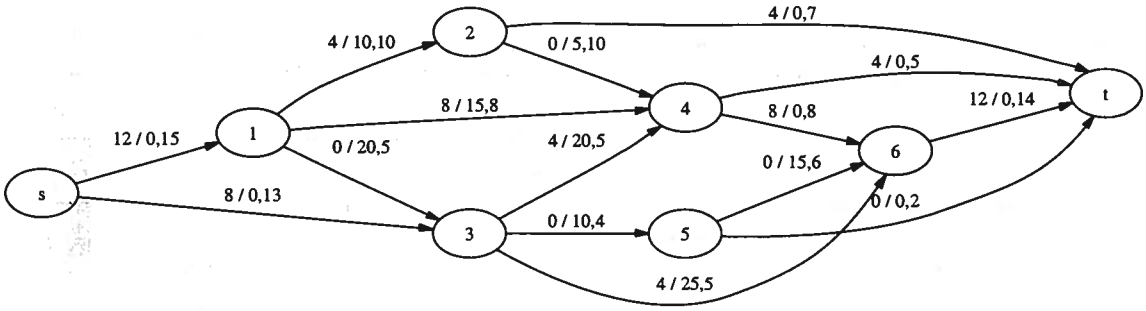

The next shortest path is $s \rightarrow 3 \rightarrow 4 \rightarrow t$, with capacity 1 and cost 20. We augment along this path:




The next shortest path is $s \rightarrow 3 \rightarrow 6 \rightarrow t$, with capacity 1 and cost 25. We augment along this path:




There are no further augmenting paths. The flow at this stage (scaled up to the original levels) is:




**Problem 2:**

a) Suppose we are given a flow network and an optimal minimum-cost circulation $F$ on the network, and we wish to change the cost of an edge $e$ in the network by one unit. We begin by computing the residual subgraph of the network. We then calculate a price function using the standard technique of adding a super-source connected to every node with zero-cost edges and computing the distance function and reduced costs. Note that (before the edge cost change) since the minimum-cost flow was optimal, there will be no negative-weight reduced-cost edges in the residual network. If the edge whose cost is being changed has non-zero reduced cost, then the minimum-cost flow does not change due to the cost adjustment: if the cost is increased, we would want to decrease the flow through the edge if possible, but it has zero flow through it. If the cost is decreased by one unit, it is still at most zero since the costs are integral, and so the flow is still an optimal minimum-cost flow on the modified network.

If the edge $e$'s reduced cost is zero, then it may have flow through it, and changing the edge cost can affect the minimum-cost flow. If we decrease the cost of $e$, then if it is not at capacity already,

we will have a negative reduced-cost edge in the residual network, which can create negative-cost cycles: this means that we need to increase the flow through $e$. If instead we increase the cost of $e$, if the edge has any flow through it, then the reverse edge in the residual network will have negative reduced cost, and can create negative-cost cycles: this means that we need to increase the flow through the reverse edge in the residual network, i.e. decrease the flow through $e$.

We can then reoptimize the flow by identifying negative-cost cycles in the residual network and augmenting along them. This is essentially the cycle-canceling algorithm. We know the result will be an optimal minimum-cost maximum flow: the circulation flow value does not change since we are only augmenting along cycles, and the final result will be an optimal minimum-cost flow since all of the negative-weight cycles will have been eliminated. This algorithm runs faster than computing the min-cost flow from scratch, since doing so would require identifying all the negative-cost cycles, while we simply need to consider the cycles that result from modifying the edge $e$, i.e. the cycles that contain $e$. that is still too much. need only $O(nm \log C)$ max-flow

3/3

**b)** We use the standard cost-scaling approach. We begin by generating the network with the highest-order bit of the costs, and performing a min-cost max-flow calculation on it. We then double the cost of each edge, which does not change the flow values in the min-cost max-flow. Next we iterate over the set of edges: for each edge $e$, if the next largest bit of $e$ is set, we adjust the cost of $e$ by 1, and use the algorithm above to recalculate its optimal min-cost flow. This requires $O(m)$ calls to the procedure above. We repeat this procedure for each bit in the costs. With $C$ as the largest cost, there are $O(\log C)$ bits in the representation of the costs, so $O(m \log C)$ calls are required.

**Problem 3:**

**a)** Suppose we are given a flow network. We can find the minimum-cost flow subject to the constraint that the flow value is at least 90% of the maximum flow value. Using a max-flow algorithm, we can find the maximum flow; call the flow value $M$. We then create a new flow network. We add a new source vertex $s^*$, and connect it to the old source $s$ with an edge of capacity $M$ and zero cost. We then connect $s$ and $t$ with an edge of capacity $M/10$ and cost zero. We then compute the minimum-cost maximum flow on this network. A flow on the original network can be extracted simply by removing the vertex $s^*$ and the edge from $s$ to $t$. The resulting flow will clearly be at least 90% of the maximum flow value. The $s$ to $t$ edge allows up to 10% of the maximum flow to be removed if it decreases the cost of the flow.

**b)** We augment the flow network in much the same way, adding a new source vertex $s^*$ connected to $s$ by a zero-cost edge with capacity $M$. We then add an edge from $s$ to $t$ with capacity $M$ and cost $K$. Then calculating a min-cost max-flow on the resulting network gives a flow that has capacity at most $M$. If it has capacity less than $M$, this means that flow was rerouted through the $s \to t$ edge. Since that edge has cost $K$, this means that passing flow through this edge reduces the flow through the rest of the graph, but it must also reduce the cost by $K$ times as much. So the desired criterion is minimized.

**Problem 4:**

**a)** Let $P = \left\{\vec{x} \ : \ A\vec{x} \leq \vec{b}\right\}$ and $Q = \{\vec{x} \ : \ D\vec{x} \leq \vec{e}\}$ be polyhedra that have empty intersection. So there exists no vector $\vec{n}$ such that $A\vec{n} \leq b$ and $D\vec{n} \leq e$. Expressing this with a block matrix, there exists no $\vec{n}$ such that

$$\vec{n}\begin{bmatrix} A & D \end{bmatrix} \leq \begin{bmatrix} b & e \end{bmatrix}$$

Applying Farkas's lemma, since there is no such $\vec{n}$, there must exist a vector $\vec{q} = \begin{bmatrix} y \\ z \end{bmatrix}$ that satisfies $\vec{q} \geq 0$ and $\vec{q}\begin{bmatrix} A & D \end{bmatrix} = 0$, i.e. $\vec{y}, \vec{z} \geq \vec{0}$ and $\vec{y}A + \vec{z}D = 0$. It also satisfies

$$\begin{bmatrix} be \end{bmatrix} \begin{bmatrix} y \\ z \end{bmatrix} \leq 0$$
$$b\vec{y} + e\vec{z} < 0$$

**b)** Suppose that polyhedra $P$ and $Q$ have empty intersection. Let $\vec{c} = \vec{y}A$. We show by contradiction that for all $x$, $\vec{c}\vec{x} < \vec{c}\vec{w}$ for all $\vec{x} \in P$ and $\vec{w} \in Q$, i.e. that there is a hyperplane separating $P$ and $Q$.

Suppose there is no such hyperplane and there exist some $\vec{x}$ and $\vec{w}$ such that

$$\vec{c}\vec{w} \leq \vec{c}\vec{x}$$

By definition of $\vec{c}$,

$$\vec{y}A\vec{w} \leq \vec{y}A\vec{x}$$

Since $\vec{x}$ is in $P$, $A\vec{x} < \vec{b}$

$$\vec{y}A\vec{w} < \vec{y}\vec{b}$$

Canceling the $y$,

$$A\vec{w} < \vec{b}$$

But by the definition of $P$, this means that $\vec{w}$ is in $P$ as well as $Q$, contradicting that they have an empty intersection. ↭

**c)** We can therefore verify quickly whether the two polyhedra $P$ and $Q$ intersect.

If they do intersect, then our proof is a point that is contained by both $P$ and $Q$. Using the definitions $P = \left\{\vec{x} \ : \ A\vec{x} \leq \vec{b}\right\}$ and $Q = \{\vec{x} \ : \ D\vec{x} \leq \vec{e}\}$, we can easily verify this.

If they do not intersect, then our proof is the $\vec{y}$ and $\vec{z}$ such that $\vec{y}, \vec{z} \geq \vec{0}$, $\vec{y}A + \vec{z}D = 0$, and $b\vec{y} + e\vec{z} < 0$. We can easily verify these claims, and by part a, they suffice to show that the two polyhedra do not intersect.

**Problem 5:**

**a)** We express the problem of currency trades to maximize the amount of yen obtainable from $D$ dollars as a linear program as follows:

Let $x_i$ be the amount of currency traded by client $i$. Then we wish to maximize

$$\sum_{\{i \ : \ b_i = ¥\}} r_i x_i$$

subject to the following constraints:

$$\begin{array}{lcll} \forall i & x_i & \geq 0 & (1) \\ \forall i & x_i & \leq u_i & (2) \\ \forall c \neq \$ & \sum_{\{i \ : \ b_i = c\}} r_i x_i - \sum_{\{i \ : \ a_i = c\}} x_i & \leq 0 & (3) \\ & \sum_{\{i \ : \ a_i = \$\}} x_i & \leq D & (4) \end{array}$$

Constraints 1 and 2 reflect the obvious constraints on how much each client can trade, and constraint 3 reflects the constraint that currency cannot be traded unless it is available to be traded. Constraint 4 ensures that at most $D$ dollars can be traded; we need not consider any currency conversions *back* to dollars and then to other currencies, because these will clearly not be optimal because directed cycles of trades do not make profits.

**b)** We now show that the currency trades resulting from the linear program can be carried out without borrowing any currency. To do so, we view the trades as a "flow" network, where each vertex represents a currency, and each directed edge represents a client that is willing to trade the source currency for the destination currency. The flow through an edge $i$ is $x_i$ from the linear program. This "flow" network cannot be considered an actual flow network because it violates some of the flow axioms; nonetheless, however, some of the network properties hold. We view the node corresponding to dollars as the source, and the node corresponding to yen as the sink. Then a path of flow between the two corresponds to a sequence of currency trades. The amount of currency traded can be found by successively dividing the amount of flow through each edge on the path by the sequence of exchange rates that converts it back into dollars and taking the minimum. No borrowing is necessary unless there is a cycle in the graph. But a cycle corresponds to a series of trades that returns to a currency it has previously traded, and since these cycles do not make a profit, it would be equally efficient to remove the cycle from the graph.

**c)** Using the same "flow" network representation as above, we can identify all paths that lead from dollars to yen, and we can find the amount of currency traded on this path by dividing by the successive exchange rates. To ensure that the resulting set of trades produces only dollars and yen, we eliminate all trades that are not on dollars-yen paths. For each edge on a dollars-yen path, we also reduce the amount of flow through that edge to the minimum required for the dollars-yen traded sequence. This gives an optimal solution that produces the maximum amount of yen and only yen and dollars.