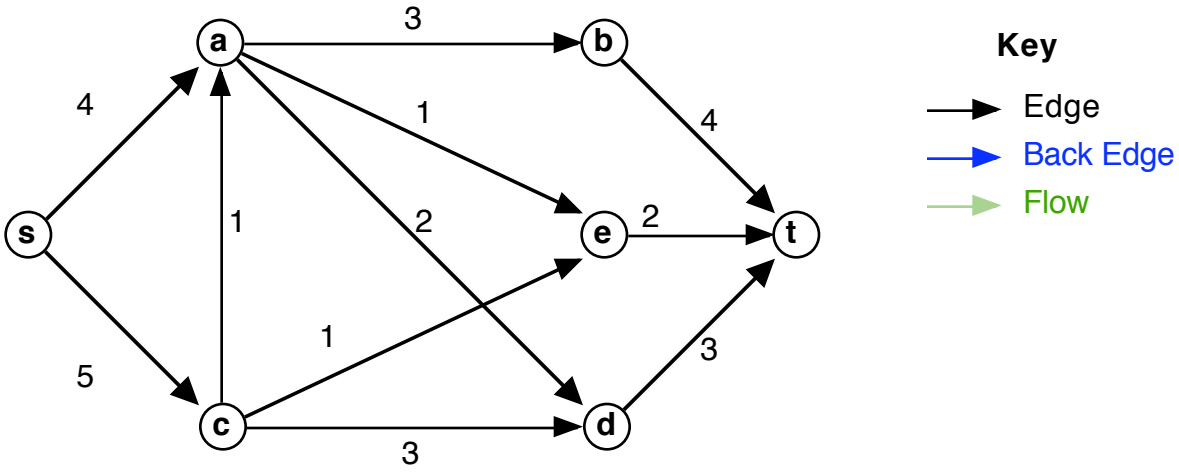
CSCI E-124-Spring, 2004 Homework 5

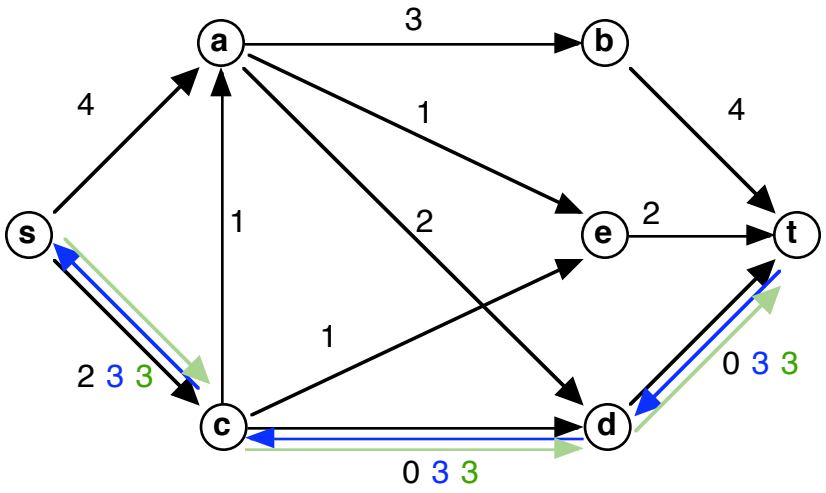
Russell Lowke, May 8th 2004

1. Find the maximum flow from *s* to *t* and the minimum cut between *s* and *t* in the network below. Show the residual network at the intermediate steps as you build the flow.

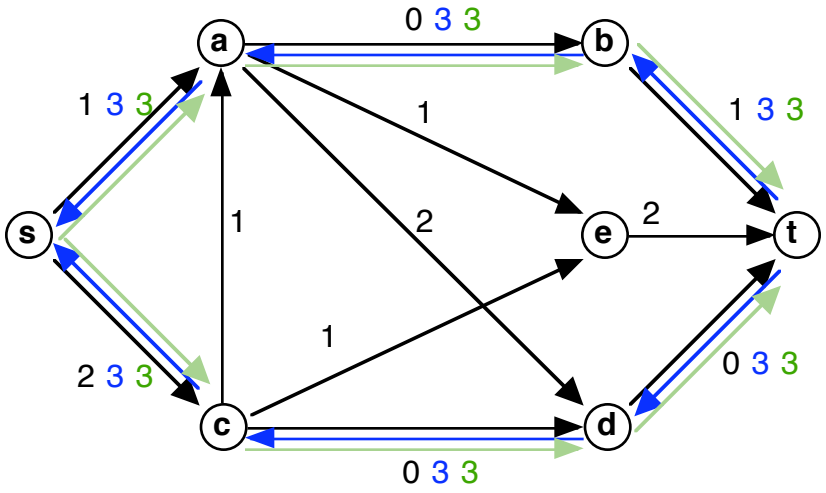
Original Network



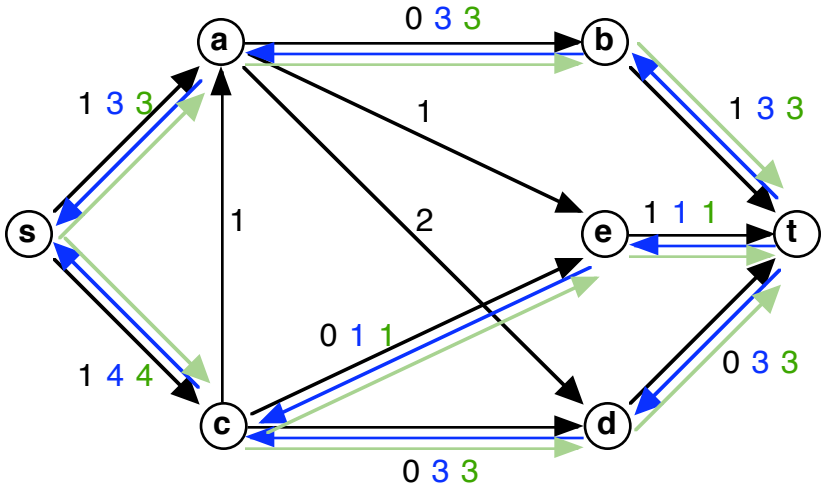
Step 1. s -> c -> d -> t for flow of 3



Step 2. s -> a -> b -> t for flow of 3
s -> c -> d -> t for flow of 3

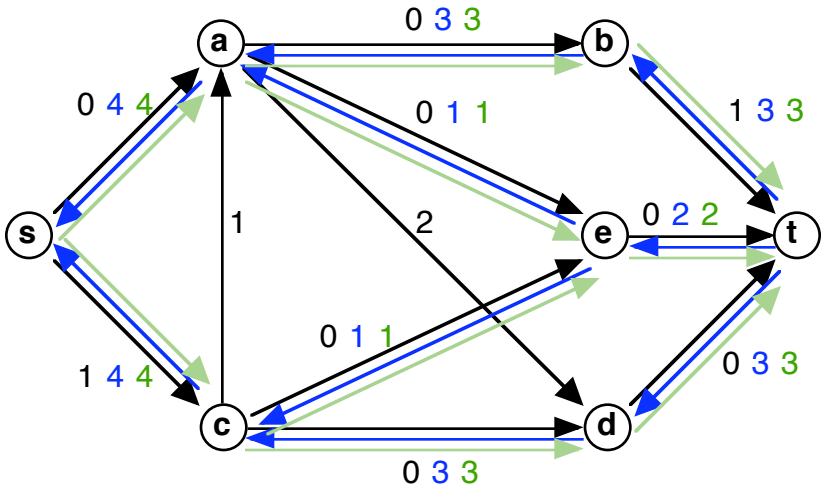


Step 3. s -> c -> e -> t for flow of 1
s -> a -> b -> t for flow of 3
s -> c -> d -> t for flow of 3



Step 4.

s -> a -> e -> t	for flow of 1
s -> c -> e -> t	for flow of 1
s -> a -> b -> t	for flow of 3
s -> c -> d -> t	for flow of 3



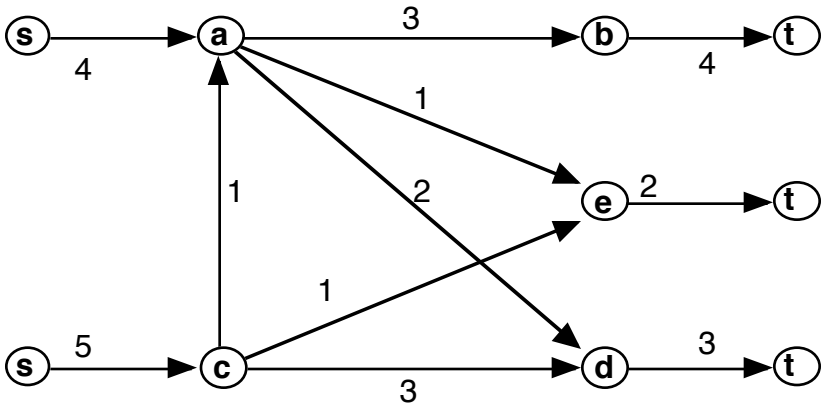
maximum flow = 1 [s -> a -> e -> t] +
1 [s -> c -> e -> t]+
3 [s -> a -> b -> t] +
3 [s -> c -> d -> t] = 8

minimum cut = V1 = { s, c, a, d }
V2 = { t, b, e}
Flow = 8.

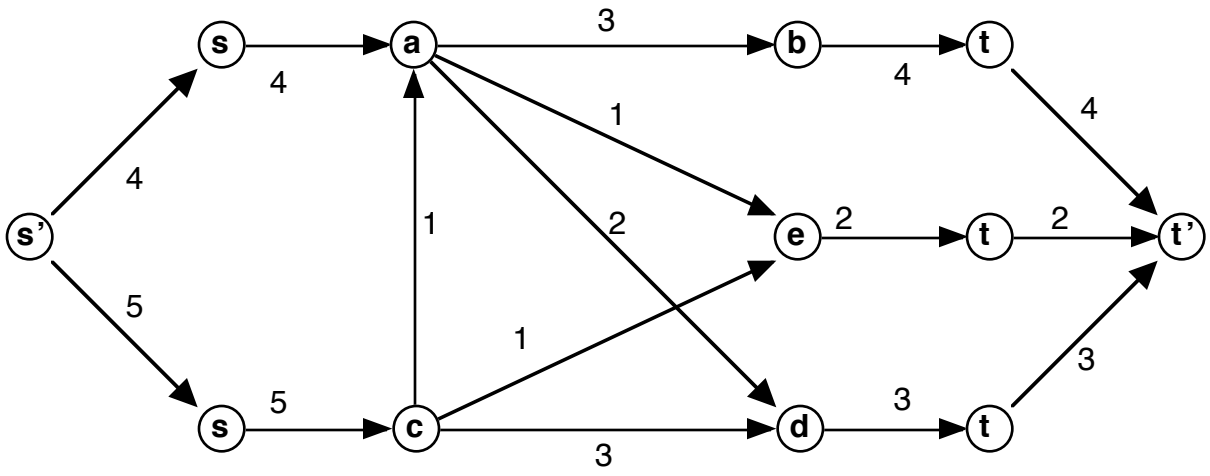
2. There are many variations on the maximum flow problem. For the two examples below, show how to solve the more general problem by reducing it to the original max-flow problem.

• There are multiple sources and multiple sinks, and we wish to maximize the flow between all sources and all sinks.

Sample Problem



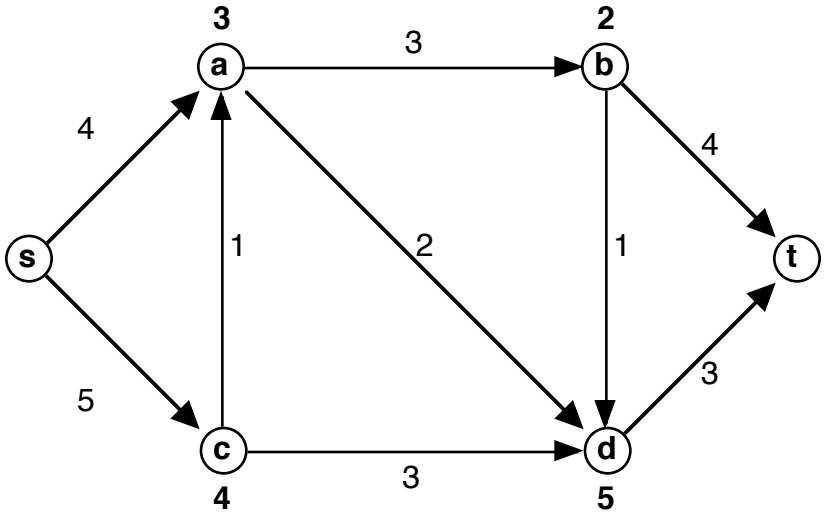
Reduction



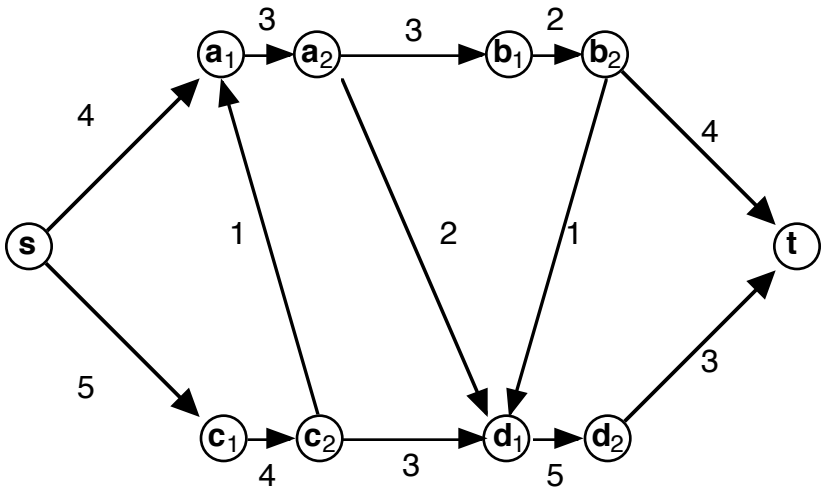
We can reduce this to the original network flow by adding in an extra starting source and an extra sink. The only requirement should be that the flow from this new source only go into the old sources, and the capacity of the edges into them cannot be less than the previous capacity out of them (so that we do not affect the max flow of the graph with our new edges). We do a similar thing for the sinks; we add a new sink and connect each old on to it with an edge with capacity greater than or equal to the capacity into the old sinks.

• Both the edges *and the vertices* (except for s and t) have capacities. The flow into and out of a vertex cannot exceed the capacity of the vertex.

Sample Problem



Reduction



We can enforce the restriction that the flow through a vertex cannot exceed a given amount by splitting the vertex into two vertices that are connected by an edge with capacity equal to the capacity of the old vertex. We connect all incoming edges to one of the new vertices, force them through the newly created edge, then connect the other new vertex to all of the outgoing edges.

For the next problem, show how to reduce the problem to linear programming.

• At each vertex, half the flow into the vertex is lost (or kept) at the vertex, and the other half flows.

In the equations given in class for reducing network flow to linear programming, there is one small change to make. When we write the equations stating that the flow into a vertex must be equal to the flow out of it, we add a factor of 2 to the outgoing flow. So the equations on every node would be that the flow into the vertex is equal to twice the flow going out of it. That is the only necessary change; the rest is a standard conversion of network flow to linear programming.

3. Consider the two-player linear program given by the following matrix. (A positive payoff goes to the row player.)

		q_1	q_2	q_3	q_4	
p_1	[	5	1	-2	-3	]
p_2	[	8	-4	-3	1	]
p_3	[	-3	-1	5	-2	]
p_4	[	-9	6	-4	9	]

• Write down the linear program to determine the row player strategy that maximizes the value of the game to the row player.

$$W_1 \leq 5p_1 + 8p_2 - 3p_3 - 9p_4$$
$$W_1 \leq 1p_1 - 4p_2 - 1p_3 + 6p_4$$
$$W_1 \leq -2p_1 - 3p_2 + 5p_3 - 4p_4$$
$$W_1 \leq -3p_1 + 1p_2 - 2p_3 + 9p_4$$

$$p_1 + p_2 + p_3 + p_4 = 1$$

Do the same for the column player.

$$W_2 \leq 5q_1 + 1q_2 - 2q_3 - 3q_4$$
$$W_2 \leq 8q_1 - 4q_2 - 3q_3 + 1q_4$$
$$W_2 \leq -3q_1 - 1q_2 + 5q_3 - 2q_4$$
$$W_2 \leq -9q_1 + 6q_2 - 4q_3 + 9q_4$$

$$q_1 + q_2 + q_3 + q_4 = 1$$

• Find an LP solver on the World Wide Web. Use the solver to solve these linear programs, and give the proper strategies for both players.

$W_1 = 0.15$ p1 = 0.26 p2 = 0.19 p3 = 0.38 p4 = 0.17

$W_2 = 0.15$ q1 = 0.23 q2 = 0.25 q3 = 0.3 q4 = 0.22

Using solver from http://riot.ieor.berkeley.edu/riot/Applications/SimplexDemo/Simplex.html

• What is the value of the game?

Value of the game is .15

Should the column player pay the row player to play, or vice versa,

The game plays in the row player's favor. The row player should pay the column player to play.

and how much should one player pay the other to make the game fair?

The row player should pay the column player 15 cents to make the game fair.

4. If we restrict the problems we look at, sometimes hard problems like counting the number of independent sets are in a graph become solvable. For instance, consider a graph that is a line on *n* vertices. (That is, the vertices are labeled 1 to *n*, and there is an edge from 1 to 2, 2 to 3, etc.) How many independent sets are there on a line graph? Also, how many independent sets are there on a cycle of *n* vertices?

A recursive definition is apparent. We can't choose two vertices that are right beside one another, so we have to skip at least one for every vertex on the line that we choose to include. So we first skip one vertex then try to choose from the rest of the list. When that is done, we skip two vertices and try the same. The number of independent sets can be found using,

IS(n) = IS(n - 2) + IS(n - 3)
IS(1) = 1
IS(2) = 2 *
IS(3) = 2

* To make sure the recursion works, this is the necessary value for two. However, the real value for two seems to be zero. I don't know why there is a conflict here, but it's necessary. Just know that, if you have two vertices connected by an edge there are zero independent sets.

• Similarly, describe how you could quickly compute the number of independent sets on a complete binary tree. Calculate the number of independent sets on a complete binary tree with 127 nodes.

The only legal values of n, being a complete binary tree, are value of the form $2^k - 1$ for some k. The strange $\left\lfloor \frac{n+1}{2} \right\rfloor 1$ is simply to get to a previous valid value of n. The logic here was to choose the root node, skip a horizontal row, then ask how many ways those sub trees can be arranged among themselves. Then leave out the root node and do it again. The recursion was:

$$T(n) = \left(T\left(\left\lfloor \frac{n+1}{2} \right\rfloor 1 \right) \right)^2$$

T(1) = 1
T(3) = 2

T(128) = 4,294,967,296

5. Consider the problem MAX-*k*-CUT, which is like the MAX CUT algorithm, except that we divide the vertices into *k* disjoint sets, and we want to maximize the number of edges between sets.

In the original algorithm, we randomly place a vertex in one of the two sets. Instead of choosing a vertex to line in one of two sets, just choose it to lie in one of k sets.

In the other local search algorithms, start with some assignment. Then, iterate through the vertices and try putting one in the opposite set and seeing if the max number of edges goes up. We still start with an assignment, yet now must check and see if shifting a vertex to *any* of the other k - 1 sets will produce a better result.

Explain how to generalize both the randomized and the local search algorithms for MAX CUT to MAX-*k*-CUT and prove bounds on their performance.

The performance for the randomized algorithm was based on the fact that, for an edge e = (u, v), the probability that the edge lay between two sets (and thus was good) was 1/2 because, of the possible placements for u and v, two of them were in the same set ((u, u) and (v, v)) and the rest would leave the edge crossing between sets which is what is wanted. Now, with k edges, the probability that the edge crosses between two sets is equal again to the 1 - the probability that it lies in one set which is k / k^2. This is because, for k sets, there are k^2 ways to assign two variables and of those k assignments put the vertices in the same set. So now we have a probability of $1 - (k / k^2)$ or just (k - 1)/k that an edge will cross between two sets. When the expectation is calculated again, we expect to get (k - 1) / k edges crossing between sets. Since the best we could get is 1, we have a bound of k / (k - 1)

For the deterministic algorithm, we know before that (when it ended) 1/2 of a given edge's neighbors will be in a different set. Now, we know that (k - 1) / k of its neighbors are in different sets. Again, this means that there will be (k - 1) / k edges that lie across sets and thus we have a bound of k / (k - 1)

6. We know that all of NP-complete reduce to each other. It would be nice if this meant that an approximation of one NP-hard problem would lead to another. But this is not the case. Consider the case of Minimum Vertex Cover, for which we have a 2-approximation. We know that (from the NP-completeness notes) the *C* is a cover in a graph *G* = (*V, E*) if and only if *V* - *C* is an independent set in *V.* Explain why this does not yield an approximation algorithm for Maximum Independent Set.

Take for example a graph of 7 vertices with a Min Vertex Cover of 3 and thus a Max Independent Set of size 4. We want to show that our 2-approximation for Min Vertex Cover yields a bounded approximation for Max Independent Set. Since it is a 2-approximation, our algorithm can return a value for the Min Vertex Cover as high as 6. This would yield an estimate for the size of the Max Independent Set of 1. This is off by a factor of *four* from the original answer. As another example, we could have a graph of 11 vertices with a Min Vertex Cover of 5 and a Max Independent Set of 6.

Our 2-approximation can yield an answer as high as 10 for the Min Vertex Cover, which corresponds to an estimate of 1 for the Max Independent Set. This is now off by a factor of 6. We can continually engineer examples in which our 2-approximation yields increasingly incorrect estimates for the Max Independent Set, so because it lacks a set bound it can not be considered a good approximation.

The Maximum Independent Set problem and the Maximum Clique problem...

An approximation algorithm in Max Clique produces an approximation algorithm in Max Independent Set because, unlike the last problem, the size of the Max Clique IS the size of the Max Independent Set in the graph G's compliment. This means that, if the size of the Max Clique is found using a 2-approximation, for example, then the size of the Max Independent Set in its compliment will also be within a factor of two (again, because they're one in the same). Contrast this with the previous question in which the answer for Min Vertex Cover was NOT an answer to Max Independent Set but gave one the information necessary to find it. For that reason, it didn't always approximate it with the same precision. Mac Clique, on the other hand, does.