The Edmonds-Karp Heuristic

Our proof of the Max-Flow-Min-Cut Theorem immediately gave us an algorithm to *compute* a maximum flow, known as the **Ford-Fulkerson algorithm**:

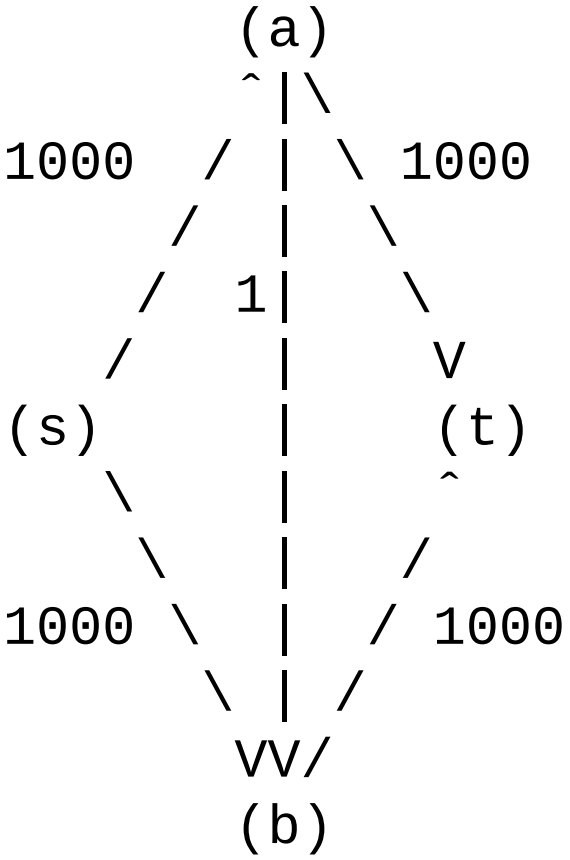
- Set f to be a zero flow.
- While the residual graph of f contains an augmenting path, find such a path, create a flow of the bottleneck capacity along this path, and add that flow to f.
- Once there is no augmenting path, return f.

Recall an important consequence of this algorithm:

Theorem: Any flow network whose capacities are all integers has a maximum flow whose edge flows are all integers.

This algorithm takes only $O(e)$ time per phase, but the number of phases is not clearly bounded by anything smaller than $|f|$. We had an example where it was that bad:

An example of bad behavior of unrestricted Ford-Fulkerson:



If we pick a first augmenting path from s to a to b to t, the resulting residual network has a path from s to b to a to t. We could then take a path through a and b, then one through b and a, and so on until we find the maximal flow of size 2000 only after 2000 phases.

It turns out that we can avoid this bad behavior by using the **Edmonds-Karp heuristic**: When we pick an augmenting path, we always pick one that is as short as possible in terms of the number of edges – so, for example, we could just pick one by breadth-first search.

Theorem: If the Edmonds-Karp heuristic is used, then the Ford-Fulkerson algorithm terminates with a maximum flow after at most ne phases.

Proof: For every vertex v, let $d(v)$ be its distance from s in the residual network. measured by the number of edges. This will change as the algorithm runs because the residual network changes. We first prove:

Lemma: As the algorithm (FF with EK) runs, $d(v)$ never decreases.

Proof of Lemma: If $d(v)$ ever decreases on some step, fix v to be a vertex where it does that is closest to s in the new residual network. The shortest path from s to v in the new residual network has length k and a last edge (u, v), and $d(u)$ did not decrease, so it was $k - 1$ in both networks.

Now if (u, v) were in the old network, $d(v)$ would have been k, but we are assuming that $d(v)$ decreased. So the augmenting path *added* (u, v) to the residual network, by killing an edge from v to u. But this path in the old network must have taken at least $k + 1$ edges to reach v plus another to reach u. This cannot be because Edmonds-Karp requires that path to reach u by the shortest route, which we know had length $k - 1$ in both networks.

Back to Proof of Theorem: We say that an edge (u, v) is **critical** for the residual flow if the augmenting path includes it and its capacity is the bottleneck capacity. So a critical edge is one that disappears from the residual network at the next step (replaced by its reversal).

We prove that between occasions when (u, v) is critical, $d(u)$ increases by at least 2. It follows that a given edge may be critical at most $n/2$ times, and thus that the total number of phases of the algorithm can be at most $(n/2)(e) = ne$ and the total running time $O(e^2n)$.

Suppose f is a flow for which (u, v) is a critical edge, and suppose that (u, v) appears *again* in the residual graph later, when the flow f' changes to the flow f''. Let k be the value of $d(u)$ in f. Since the augmenting path in f is a shortest path and goes through u and v, we know that $d(v) = k + 1$ in f. The augmenting path in f', that puts (u, v) back in the residual graph, must include the edge (v, u). Since it is also a shortest path, and $d(v)$ is still at least $k + 1$, we know that $d(u)$ at this point must be at least $k + 2$.

Network-Flow Applications

The Max-Flow-Min-Cut Theorem and the existence of integer-size flows give easy proofs of some classic results in graph theory:

Hall's Theorem: (early 1900's) A bipartite graph $G = (U, V, E)$, with $|U| = |V| = n$, has a perfect matching iff there does *not* exist a set $A \subseteq U$ such that the set $\Gamma(A) = \{v \in V : \exists u \in A : (u, v) \in E\}$ is smaller than A.

Proof: Set up a flow network with a node s that has edges of weight 1 to every node in U and a node t that has edges of weight 1 *from* every node in V. Give every edge in E weight 1. If this network has a flow of size n, then it has an integer flow, which must have edge values of 0 or 1, and the edges of E used in this flow form a perfect matching. If the network does not have such a flow, there must be a cut (X, Y) of capacity $n-1$ or smaller. The set X contains s plus some nodes of U and V. (Why must it contain at least one node of U?). Let A be $X \cap U$. We will show that $|\Gamma(A)| < |A|$.

We first modify the cut by taking any nodes in $\Gamma(A) \cap Y$ and moving them to X. This doesn't *increase* the capacity of the cut – if y is such a vertex then the edge (y, t) is the only one that now crosses the cut but didn't before, and at least one edge from A to y no longer crosses the cut.

Let $|A| = k$. The $n - k$ edges from s to $U \setminus A$ cross the cut, as do the edges from $\Gamma(A)$ to t. This gives us at least $n - k + |\Gamma(k)|$ edges crossing the cut, so if $\Gamma(A) \geq k$ we have a contradiction because the capacity of the cut was assumed to be at most $n - 1$.

Here is another classic result in graph theory:

Menger's Theorem: (1927) In any directed graph with nodes s and t, the maximum number of edge-disjoint paths from s to t is equal to the minimum number of edges whose removal separates s from t.

Proof: This is simply the special case of the Max-Flow-Min-Cut Theorem when all the edges have weight 1. If the size of the minimum cut is k, there must exist a flow of size k, and hence an integer flow of size k. We can easily divide this flow into k edge-disjoint paths from s to t.

The Bipartite Matching Problem

Recall that a graph is **bipartite** if its vertex set V is partitioned into two sets X and Y and the edge set E is a subset of $X \times Y$. A **matching** is a subset of E where no two distinct edges share a vertex. A matching is **perfect** if $|X| = |Y| = n$ and the matching has n edges.

We saw earlier that although a greedy algorithm only gets a *maximal* matching in a bipartite graph, we could get a *maximum* matching in polynomial time by another approach. We also saw that the maximum bipartite matching was a special case of finding an element of maximum cardinality in the **intersection of two matroids**, and we asserted without proof that this latter problem can be solved in polynomial time.

We can also solve the bipartite matching problem in polynomial time by mapping it to a network flow problem. Given a bipartite graph $G = (X, Y, E)$, our network flow diagram consists of the following vertices and edges, where all edges have weight 1:

- A source vertex s and a sink vertex t,
- Copies of X and Y
- An edge from s to each member of X,
- The edges of E from X to Y, and
- An edge from each vertex of Y to t.

The Ford-Fulkerson algorithm gives us a maximum flow, with integer edge flows, in polynomial time. If m is the size of this flow, we must have exactly m edges of E with unit flow, and zero flow on the rest of E. Since only one unit of flow can enter any vertex in X or leave any vertex in Y, these m edges cannot share a vertex on either side and so constitute a matching.

Furthermore, any matching in G can be converted into a flow of this kind, so finding a maximum flow corresponds exactly to finding a maximum matching in G. A perfect matching exists in a graph with $|X| = |Y| = n$ iff the maximum flow has size n.

Scheduling problems often involve finding matchings, and can often be modeled as network flow problems. Consider the following problem from last year's 611 midterm:

- We have m workers and n shifts that need to be covered.
- Each worker w_i has a number t_i of shifts that she will work.
- Each shift d_j has a number s_j of workers needed on that shift.
- Fortunately, $\Sigma_i\, t_i = \Sigma_j\, s_j = r$.
- Each worker w_i has a list L_i of preferred shifts.
- We have a limit c on the number of non-preferred shifts we may assign in all.

We can model the sets L_i as a bipartite graph, where vertex w_i is connected to each element of L_i. Our ideal goal would be to find a set of edges in this graph that has exactly t_i edges including each vertex w_i, and exactly s_j edges touching each edge d_j. If all the s_i's and t_j's were 1, this would be the perfect matching problem.

It is easy to model this situation as a flow network, much as we did for the matching problem. We have a source vertex s with an edge of capacity t_i to each vertex w_i. We represent the sets L_i by edges of capacity 1, and have an edge of capacity s_j from each vertex d_j to the sink vertex t.

An integer flow in this network is a partial assignment of workers to shifts that respects the workers' preferences and does not exceed any worker's supply or any shift's demand. If we have a flow of size n, we can complete the entire assignment with every worker getting a preferred shift. If we have a flow of size $n - c$, then we can make $n-c$ preferred assignments and then assign the remaining workers to the remaining shifts arbitrarily, still meeting the constraint that at most c assignments may be non-preferred.

The Baseball Elimination Problem

Here is another application of network flow taken from the Kleinberg-Tardos textbook. Near the end of a baseball season, fans want to know whether their team has been **eliminated**, meaning that they cannot possibly finish in first place. (For our purposes here, we will define "first place" as having *at least as many* wins as any other team in the division. Under current rules, teams tying for first place have a playoff game unless all would qualify for post-season play anyway.)

The simplest way to argue that Team X has been eliminated is to prove that some other team Y will finish ahead of X using the **magic number method**. If the sum $W_Y + L_X$ of Y's wins and X's losses *exceeds* the total number of games played (which is the same for all teams), then even if X win all their remaining games Y will still have more wins.

But there are situations where X has no chance of finishing first, but the magic number method does not prove that X has been eliminated. Suppose that with one game left for each team, Y and Z are tied for the lead and X is one game behind (has one fewer win). The magic number method says that $W_Y + L_X$ and $W_Z + L_X$ are each *equal* to the total number of games played, so there is still a chance for X to tie.

But what if Y and Z play their last game *against each other*? Then the winner of the game will have two more wins than X has now, and X cannot possibly have more than one more win.

We can formalize this argument as follows: X now has k wins and can finish with at most $k + 1$ wins. The *set* of teams $S = \{Y, Z\}$ now has $2k + 2$ wins and must have another win from the final game between X and Y. By the Lake Wobegon Principle ("it is not possible for all elements of a set of numbers to be strictly below average"), some team in S must finish with *more than* $(2k+3)/2$ wins, which is to say with at least $k+2$, strictly more than X could have.

A natural question is whether we need to consider even more complicated arguments to determine whether a team has been eliminated. We'll use our knowledge of the network flow problem to prove:

Theorem: If X has been eliminated (cannot finish first under any possible scenario of results for the remaining games) then there exists a set of teams S such that we can prove as follows that at least one team in S must finish ahead of X. Specifically, if k is the best X could do (current wins plus future games) then the wins of teams in S plus the number of games internal to S, divided by the number of teams in S, is greater than k.

Proof: We set up a network flow problem, with:

- A source vertex s, with edges to
- A column of vertices for every *pair* of non-X teams – the weight of the edge from s to "Y versus Z" is the number of games left between Y and Z,
- A column with a vertex for each team, and an edge from each "Y versus Z" to both Y and Z, and finally
- A sink vertex t, and an edge from each team Y to t whose capacity is $k - W_Y$, where W_Y is the number of wins Y currently has.

We have set up a flow network in which the integer flows represent scenarios for the remainder of the season. If we have a maximum possible flow, equal to g, the number of games remaining that don't involve X, then each game has a winner (actually the flow into "Y versus Z" splits into an integer number of X wins and an integer number of Y wins). Also, no non-X team exceeds k total wins. Thus a scenario exists in which X at least ties for the lead.

What if there is *no* flow of the maximum possible size g? Then we know that X is eliminated, and it remains to show that a set S exists that satisfies the conditions of the theorem.

By the Max-Flow-Min-Cut Theorem, there must be a cut of our flow network whose capacity is at most $g - 1$. We first show that without loss of generality, this cut has a set S of vertices in the team column on the left side, together with the vertices in the pair column that represent pairs contained to S. If our original cut has a pair vertex on the left that has one or both of its edges across the boundary, we can move this vertex to the right. The capacity of the cut then either stays the same or goes down by one.

Our revised cut has s, the pairs from S, and the teams in S on the left, and all other vertices on the right. What is the capacity of this cut?

- From s to the pairs column, we have the number of games left that are not internal to S, which is g minus the number internal to s.
- From the pairs column to the teams column, we have arranged that no edges cross the cut.
- From the teams column to t, we have the sum over teams Y in S of $k - W_Y$.

The total is g, minus the number of games internal to S, plus the sum over S of $k - W_Y$. We are given that this is at most $g - 1$, from which it follows that the number of games internal to S is *strictly greater* than the sum of $k - W_Y$. Thus if we add W_Y for each Y in S, and add in the number of games internal to S, we get *strictly more* than $k|S|$. Therefore the average number of wins among the teams in S at the end of the season will be greater than k, and we have a proof that X is eliminated.