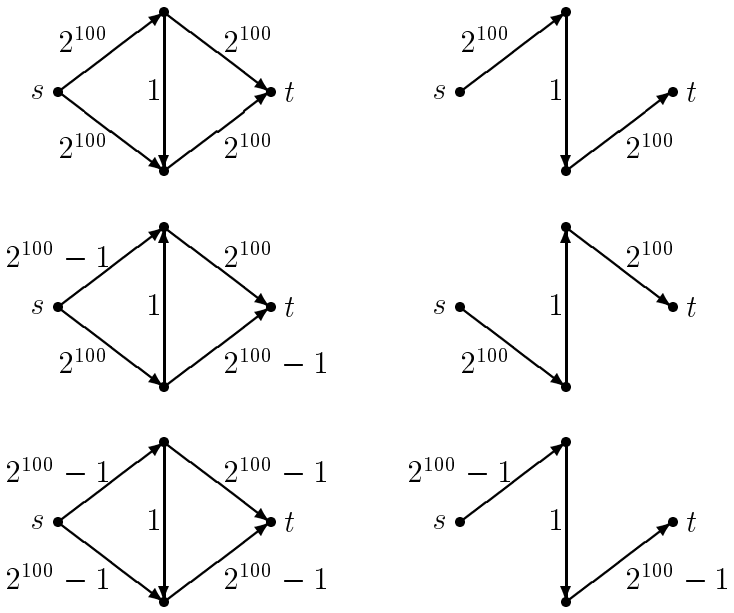
Lecture 17 More on Max Flow

The Max Flow-Min Cut Theorem gives an algorithm for finding a flow with maximum value in a given network as long as the capacities are rational numbers. This algorithm was first published in 1956 by Ford and Fulkerson [34].

The algorithm works as follows. We begin with the zero flow, then repeatedly find an augmenting path p and push d additional units of flow along p from s to t, where $d > 0$ is the bottleneck capacity of p (minimum edge capacity along p). We continue until it is no longer possible to find an augmenting path, *i.e.* until the residual graph has no path from s to t. We know at that point by the Max Flow-Min Cut Theorem that we have a max flow.

If the edge capacities are integers, this algorithm increases the flow value by at least 1 with each augmentation, hence achieves a maximum flow after at most $|f^*|$ augmentations. Moreover, each augmentation increases the flow by an integral amount, so $|f^*|$ is an integer. Unfortunately, $|f^*|$ can be exponential in the representation of the problem, and the algorithm can run for this long if the augmenting paths are not chosen with some care.

Example 17.1 The following diagram illustrates the first few augmentations in a flow problem with large capacities. The residual graphs are shown on the left-hand side and the augmenting paths on the right. This sequence of augmentations will take 2^{101} steps to converge to a max flow, which has value 2^{101}.



□

In fact, if the capacities are irrational, the process of repeated augmentation along indiscriminately chosen augmenting paths may not produce a max flow after a finite time, as the following example shows.

Example 17.2 Let r be the positive root of the quadratic $x^2 + x - 1$:

$$r \;=\; \frac{-1+\sqrt{5}}{2} \;\approx\; .618\ldots$$

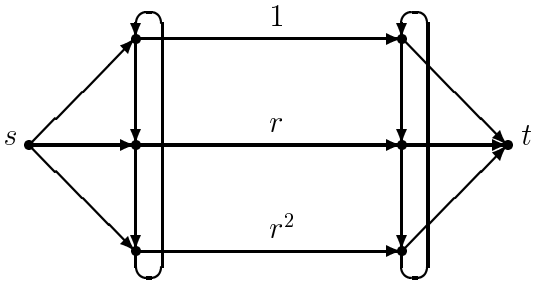
Then $r^2 = 1 - r$, and more generally, $r^{n+2} = r^n - r^{n+1}$ for any $n \geq 0$. Also, since $0 < r < 1$,

$$1 \;>\; r \;>\; r^2 \;>\; r^3 \;>\; \cdots \;>\; 0\ .$$

Note

$$r + 2 \;=\; \frac{1}{1-r} \;=\; \sum_{n=0}^{\infty} r^n\ .$$

Consider the following flow network:



The three horizontal interior edges (call them the *flumes*) have the capacities shown, and all other edges have capacity $r + 2$. The max flow value is $1 + r + r^2 = 2$, since this is the minimum cut capacity obtained by cutting the flumes; any other cut has capacity at least $r + 2 > 2$.

Suppose that in the first augmenting step, we push one unit of flow directly from s to t along the top flume. This leaves residual capacities of $0, r$, and r^2 on the flumes.

Now we perform the following loop, which after n iterations will result in the flumes having residual capacities 0, r^{n+1}, and r^{n+2} in some order: choose the flume with minimum nonzero residual capacity, say d, and push d units of flow from s forward along that flume, back through the saturated flume, and then forward through the remaining flume to t. Suppose that we start with residual capacities $0, r^n$, and r^{n+1} on the flumes. The minimum nonzero residual capacity is r^{n+1}, and the new residual capacities will be r^{n+1}, $r^n - r^{n+1} = r^{n+2}$, and 0, respectively. The situation is the same as before, only rotated.

The loop can be repeated indefinitely, leaving ever higher powers of r on the flumes. We always have sufficient residual capacity on the non-flumes. The residual capacities tend to 0, so the flow value tends to the maximum flow value 2.

With irrational capacities, the sequence of augmentations need not even converge to the maximum flow value. An example of this behavior can be obtained from the graph above by adding an edge (s, t) of weight 1. The same infinite sequence of augmentations converges to a flow of value 2, but the maximum flow value is 3. □

17.1 Edmonds and Karp's First Heuristic

Edmonds and Karp [30] suggested two heuristics to improve this situation. The first is the following:

Always augment by a path of maximum bottleneck capacity.

Definition 17.3 A *path flow* in G is a flow f that takes nonzero values only on some simple path from s to t. In other words, there exist a number d and a simple path $u_0, u_1, \ldots, u_k$ with $s = u_0$, $t = u_k$, and such that

$$\begin{aligned} f(u_i, u_{i+1}) &= d, \quad 0 \le i \le k-1 \\ f(u_{i+1}, u_i) &= -d, \quad 0 \le i \le k-1 \\ f(u, v) &= 0, \quad \text{for all other } (u, v). \end{aligned}$$

□

Lemma 17.4 *Any flow in G can be expressed as a sum of at most m path flows in G and a flow in G of value 0, where m is the number of edges of G.*

Proof. Let f be a flow in G. If $|f| = 0$, we are done. Otherwise, assume $|f| > 0$ (the argument for $|f| < 0$ is symmetric, interchanging the roles of s and t). Define a new capacity function $c'(e) = \max\{f(e), 0\}$ and let G' be the graph with these capacities. Then f is still a flow in G', and since $c' \leq c$, any flow in G' is also a flow in G. By the Max Flow-Min Cut Theorem, the null flow in G' must have an augmenting path, which is a path from s to t with positive capacities; by construction of G', every edge on this path is saturated by f. Take p to be the path flow on that path whose value is the bottleneck capacity. Then the two flows p and $f - p$ are both flows in G', and at least one edge on the path (the bottleneck edge) is saturated by p.

Now we repeat the process with $f - p$ to get $c'' \leq c'$ and G'', and so on. Note that G'' has strictly fewer edges than G', since at least the bottleneck edge of p has disappeared. This process can therefore be repeated at most m times before the flow value vanishes. The original f is then the sum of the remaining flow of value 0 and the path flows found in each step. □

We now consider the complexity of maximum-capacity augmentation.

Theorem 17.5 *If the edge capacities are integers, then the heuristic of augmentation by augmenting paths of maximum bottleneck capacity results in a maximum flow f^* in at most $O(m \log |f^*|)$ augmenting steps.*

Proof. By Lemma 17.4, f^* is a sum of at most m path flows and a flow of value 0, therefore one of the path flows must be of value $|f^*|/m$ or greater. An augmenting path of maximum bottleneck capacity must have at least this capacity. Augmenting by such a path therefore increases the flow value by at least $|f^*|/m$, so by Lemma 16.7(d) of the previous lecture, the max flow in the residual graph has value at most $|f^*| - |f^*|/m = |f^*|(\frac{m-1}{m})$. Thus after k augmenting steps, the max flow in the residual graph has value at most $|f^*|(\frac{m-1}{m})^k$. Hence the number of augmenting steps required to achieve a max flow is no more than the least number k such that

$$|f^*|(\frac{m-1}{m})^k \quad < \quad 1 \ .$$

Using the estimate

$$\log m - \log(m-1) \quad = \quad \Theta(\frac{1}{m}) \ , \tag{24}$$

we obtain $k = \Theta(m \log |f^*|)$. The estimate (24) follows from the limit

$$\lim_{n \to \infty} (1 - \frac{1}{n})^n \quad = \quad \frac{1}{e} \ .$$

□

Finding a maximum capacity augmenting path can be done efficiently using a modification of Dijkstra's algorithm (Homework 5).

17.2 Edmonds and Karp's Second Heuristic

The method described above is still less than completely satisfactory, since the complexity depends on the capacities. It would be nice to have an algorithm whose asymptotic worst-case complexity is a small polynomial in m and n alone.

The following algorithm produces a max flow in time independent of the edge capacities. This algorithm is also due to Edmonds and Karp [30]. It uses the following heuristic to achieve an $O(m^2 n)$ running time:

Always choose an augmenting path of minimum length.

Definition 17.6 The *level graph* L_G of G is the directed breadth-first search graph of G with root s with sideways and back edges deleted. The *level* of a vertex u is the length of a shortest path from s to u in G. □

Note that the level graph has no edges from level i to level j for $j \geq i+2$. This says that any shortest path from s to any other vertex is a path in the level graph. Any path with either a back or sideways edge of the breadth-first search graph would be strictly longer, since it must contain at least one edge per level anyway.

Lemma 17.7 *(a) Let p be an augmenting path of minimum length in G, let G' be the residual graph obtained by augmenting along p, and let q be an augmenting path of minimum length in G'. Then $|q| \geq |p|$. Thus the length of shortest augmenting paths cannot decrease by applying the above heuristic.*

(b) We can augment along shortest paths of the same length at most $m = |E|$ times before the length of the shortest augmenting path must increase strictly.

Proof. Choose any path p from s to t in the level graph and augment along p by the bottleneck capacity. After this augmentation, at least one edge of p will be saturated (the bottleneck edge) and will disappear in the residual graph, and at most $|p|$ new edges will appear in the residual graph. All these new edges are back edges and cannot contribute to a shortest path from s to t as long as t is still reachable from s in the level graph. We continue finding paths in the level graph and augmenting by them as long as t is reachable from s. This can occur at most m times, since each time an edge in the level graph disappears. When t is no longer reachable from s in the level graph, then any augmenting path must use a back or side edge, hence must be strictly longer. □

This gives rise to the following algorithm:

Algorithm 17.8 (Edmonds and Karp [30]) Find the level graph L_G. Repeatedly augment along paths in L_G, updating residual capacities and deleting edges with zero capacity until t is no longer reachable from s. Then calculate a new level graph from the residual graph at that point and repeat. Continue as long as t is reachable from s.

With each level graph calculation, the distance from s to t increases by at least 1 by Lemma 17.7(a), so there are at most n level graph calculations. For each level graph calculation, there are at most m augmentations by Lemma 17.7(b). Thus there are at most mn augmentations in all. Each augmentation requires time $O(m)$ by DFS or BFS, or $O(m^2n)$ in all. It takes time $O(m)$ to calculate the level graphs by BFS, or $O(mn)$ time in all. Therefore the running time of the entire algorithm is $O(m^2n)$.