Choosing augmenting paths in the Ford-Fulkerson algorithm

Vassos Hadzilacos

In what follows we assume that all edge capacities are non-negative integers. We have seen that in the Ford-Fulkerson algorithm, if we choose augmenting paths arbitrarily, it is possible that we will perform \( C \) augmenting steps before the algorithm terminates, where \( C \) is the sum of the capacities of the edges out of the source. If capacities are large, this can lead to very poor performance.

We will now see that by choosing augmenting paths more wisely, we can improve the performance of the algorithm dramatically.

In what follows \( n \) is the number of nodes and \( m \) is the number of edges in the flow graph. It is reasonable to assume that every node is reachable from \( s \) (unreachable nodes are not used in any \( s \rightarrow t \) flow and can be deleted from the flow graph, along with their incoming and outgoing edges). Note that under this assumption \( m \geq n - 1 \) and so \( n = O(m) \).

Choosing a “widest” path

Let \((G, s, t, c)\) be a flow graph with integer capacities, and \(f\) be a flow in this flow graph. Consider the residual graph \(G_f\). We define the width of an \(s \rightarrow t\) path \(p\) in \(G_f\) to be the minimum residual capacity of any edge on \(p\). Note that this is the amount by which the flow \(f\) is improved if we augment it along \(p\). Instead of picking an arbitrary augmenting path, we will choose an augmenting path of maximum width — i.e., a path that improves the value of the flow as much as is possible in a single augmenting step.

This can be done by using a simple variation of Dijkstra’s algorithm. So, we can find the widest augmenting path in \(O(m \log n)\) time.

It turns out that if we always choose the augmenting path in this way, the Ford-Fulkerson algorithm will finish in at most \(O(m \log C)\) iterations.

Choosing a path with fewest edges

Another way to choose augmenting paths to ensure good performance of the Ford-Fulkerson algorithm is to always use as augmenting path one with the fewest edges. We can find such a path in linear time in the size of the graph using Breadth-First Search (starting at \(s\)). So, that’s even faster (and easier) than finding the widest path.

It turns out that with this choice of augmenting paths, the Ford-Fulkerson algorithm is guaranteed to terminate after at most \(mn\) augmentations.

In what follows, when we speak of “shortest path” we mean a path with the minimum number of edges (as opposed to the minimum sum of capacities, as in the weighted shortest path problem).

The basic intuition behind this fact is as follows. Each augmenting step causes at least one edge to be deleted from the residual graph. It may also cause some edges to be inserted. The deleted edges make shortest paths longer. And, it turns out, that re-inserted edges do not create shorter paths than the already existing ones. Thus, shortest paths to nodes get longer and longer. Since a shortest \(s \rightarrow t\) path cannot have more than \(n - 1\) edges, after some number of augmenting steps \(s\) and \(t\) will become disconnected, meaning that the FF algorithm terminates. A careful counting argument shows that this must happen after no more than \(mn\) augmenting steps. We now give the detailed proof that makes this intuition precise.

Let \(G_k\) be the residual graph after \(k\) augmenting steps, using the shortest path method. Define the depth of a node \(u\) in \(G_k\) to be the minimum number of edges on any path from \(s\) to \(u\) in \(G_k\) (i.e., the
depth of $u$ on a BFS tree of $G_k$ rooted at $s$); we denote this as $\text{depth}_k(u)$. Consider how augmenting step $k + 1$ changes the residual graph $G_k$ to $G_{k+1}$. Let $p = u_0, u_1, \ldots, u_m$ be the augmenting path of $G_k$ used in this step, where $u_0 = s$ and $u_k = t$, and each node $u_i$ has depth $i$ in $G_k$; this is because, by the choice of augmenting paths, $p$ is a shortest path from $s$ to $t$. For each edge $(u_i, u_{i+1})$ on $p$:

- If $(u_i, u_{i+1})$ is a forward edge, this edge is deleted from the residual graph if and only if the augmenting step saturated $(u_i, u_{i+1})$ in the flow graph; and the reverse edge $(u_{i+1}, u_i)$ is added to the residual graph as a backward edge if and only if the edge $(u_i, u_{i+1})$ had zero flow before the augmenting step.

- If $(u_i, u_{i+1})$ is a backward edge, this edge is deleted from the residual graph if and only if the augmenting step removed all flow from $(u_{i+1}, u_i)$. Since $(u_{i+1}, u_i)$ is a backward edge, this edge is deleted from the residual graph if and only if the augmenting step saturated $(u_i, u_{i+1})$.

No other changes to the set of edges of the residual graph are made by the augmenting step. So, every edge added to the residual graph during augmenting step $k + 1$ goes from a node with depth $i + 1$ to a node with depth $i$ in $G_k$. Thus, we have:

**Claim 1.** If an edge $(u, v)$ is added to the residual graph $G_{k+1}$ during augmenting step $k + 1$ (i.e., $(u, v)$ is not in $G_k$ but is in $G_{k+1}$), then $\text{depth}_{k+1}(u) = \text{depth}_k(v) + 1$.

Next we will show that the depth of a node does not increase as a result of an augmenting step.

**Claim 2.** For every node $u$, $\text{depth}_{k+1}(u) \geq \text{depth}_k(u)$.

**Proof.** Let $u$ be any node, and let $u_1, u_2, \ldots, u_m$, where $u_1 = s$ and $u_m = u$ be a shortest (fewest edges) path from $s$ to $u$ in $G_{k+1}$. By definition of depth, $\text{depth}_{k+1}(u_i) = i$ for each $i, 1 \leq i \leq m$.

By induction we show that, for each $i, 1 \leq i \leq m$, $\text{depth}_k(u_i) \leq \text{depth}_{k+1}(u_i)$. This is clearly true for the basis $i = 1$, since $u_1 = s$ and $s$ is has depth 0 in every residual graph. Suppose, it is true for $u_i$, for some $i, 1 \leq i < m$; we will prove that it is true for $u_{i+1}$. There are two cases: either edge $(u_i, u_{i+1})$ is in $G_k$, or it was added to the residual graph in augmenting step $k + 1$.

**Case 1.** $(u_i, u_{i+1})$ is in $G_k$. In this case, we have:

$$\text{depth}_k(u_{i+1}) \leq \text{depth}_k(u_i) + 1 \leq \text{depth}_{k+1}(u_i) + 1 = i + 1$$

**Case 2.** $(u_i, u_{i+1})$ is not in $G_k$ but is added to $G_{k+1}$. In this case, we have:

$$\text{depth}_k(u_{i+1}) \leq \text{depth}_k(u_{i+1}) + 1 = \text{depth}_k(u_i) \leq \text{depth}_{k+1}(u_i) = i + 1$$

In each augmenting step, at least one edge is deleted from the residual graph, namely every edge on the augmenting path that has minimum residual capacity $b$. Let $(u, v)$ be an edge that is deleted in augmenting step $\ell$, and let $i = \text{depth}_\ell(v)$. Since $(u, v)$ is on an augmenting path, which is a shortest $s \to t$ path, $\text{depth}_\ell(v) = i - 1$. If $(u, v)$ is added to the residual graph as a result of a subsequent augmenting step $k + 1 > \ell$, by Claim 1, $\text{depth}_k(u) = \text{depth}_k(v) + 1$. And since by Claim 2 the depth of a node does not decrease in successive augmentations, $\text{depth}_{k+1}(v) \geq \text{depth}_k(v)$. So, $\text{depth}_{k+1}(u) = \text{depth}_k(v) + 1 \geq \text{depth}_\ell(v) + 1 = i + 1$. Thus, the depth of node $u$ must have increased by at least 2 (from
between an augmenting step \( \ell \) that deleted \((u, v)\) from the residual graph and the subsequent augmenting step \( k+1 \) that reinserted it. Since the minimum depth of a node is 0 and the maximum is \( n-1 \), an edge can be added at most \( n/2 \) times to the residual graph. Since there are at most \( 2m \) edges in the residual graph, and each augmenting step deletes at least one edge, after \( 2m(n/2) = mn \) augmenting steps all edges will have been deleted and cannot be inserted (because each of them will have been re-inserted the maximum number of times that it can). Thus, the Ford-Fulkerson algorithm terminates after at most \( mn \) augmenting steps if at each step we chose to augment the flow along a shortest \( s \rightarrow t \) path in the residual graph.

**Summary of Ford-Fulkerson algorithm running time**

Recall that each iteration of the Ford-Fulkerson algorithm (augmentation step) takes \( O(m) \) time. So,

- With arbitrary choice of augmenting paths, the algorithm requires at most \( C \) augmentation steps, so it takes \( O(mC) \) time. This is a *pseudo polynomial-time algorithm*: its running time is a polynomial of the graph size and the values (not the size) of the capacities.

- By always choosing a maximum-width augmenting path, the algorithm requires at most \( m \ln C \) augmentation steps. Each augmenting step now takes \( O(m \log n) \) time (not just \( O(m) \), because finding the widest augmenting path takes \( O(m \log n) \), not \( O(m) \), time). So the entire FF algorithm takes \( O(m^2 \log n \log C) \) time. This is a *polynomial-time algorithm*: its running time is a polynomial of the graph size and the size of the (binary representation of the) capacities.

- By always choosing an augmenting path with the fewest edges, the algorithm requires at most \( mn \) augmentation steps, so it takes \( O(m^2 n) \) time (now finding a shortest augmenting path takes only \( O(m) \) time). This is a *strongly polynomial-time algorithm*: assuming we can do arithmetic operations on capacities in \( O(1) \) time, its running time is a polynomial of the graph size only, and independent of the capacities. (Doing arithmetic operations on capacities in constant time is not an unreasonable assumption: typically capacities are small enough that they can be represented as 64-bit integers.)