Lecture Notes on the Edmonds/Karp algorithm — CS 570

David Kempe

November 24, 2004

The Edmonds/Karp algorithm is a specific implementation of the generic Ford/Fulkerson algorithm for computing a Maximum Flow in a network. Recall that the Ford/Fulkerson algorithm looks as follows:

Algorithm 1 Ford Fulkerson

1: Start with flow $f_e = 0$ for all edges e.
2: **while** the residual graph G_f contains an s-t path P **do**
3: Augment the flow f along some such path P.
4: **end while**

As we saw in class, and the textbook explains, this algorithm is always correct, and will always terminate (with an integral Max-Flow) within $O(C)$ iterations when all edge costs are integers; here, $C \leq \sum_e c_e$. However, a poor choice of the path P for augmentation may result in actually taking $\Omega(C)$ iterations — furthermore, when the edge costs are irrational, the algorithm may not terminate at all. Hence, we want to make "good" choices of the path P for augmentation.

A first natural choice is to augment along a path P with largest bottleneck capacity, called a *widest path*. We did not analyze this heuristic in class, nor will we here in detail, but the following can be proved about it.

Theorem 1 *If the widest path is chosen in each iteration, then the Ford/Fulkerson algorithm terminates in $O(m \log C)$ iterations.*

Notice that this is actually polynomial in the size of the input, as opposed to the previous bound, which was pseudo-polynomial.

Proof Sketch. The idea is that each flow can be "decomposed" into at most m paths: that is, we can identify at most m paths P_i from s to t, carrying flow, such that these paths together account for all of the flow f. By the Pigeon Hole Principle, one of them carries at least a $1/m$ fraction of the total flow, so the widest path does as well. Thus, if F^* is the value of a maximum flow, then in each iteration, we increase the amount of flow we found by a $1/m$ fraction of $F^* - \nu(f)$. Some relatively simple arithmetic now shows that after $O(\log F^* \log m)$ iterations, we have found a flow of value F^*, i.e., a maximum flow. ■

The other important question is of course whether we can actually find the widest path. As observed in class, a relatively simple dynamic program lets us do that. Perhaps even simpler is a divide-and-conquer approach. We try to find the "bottleneck edge" in the residual graph G_f via binary search. Having sorted the edges by their residual capacity c'_e, we start with the median edge e, and ignore all edges of smaller capacity. If there is an s-t path using only the remaining edges, we know that the bottleneck edge of the widest path has capacity at least c'_e, and can continue recursively with the edges of capacity at least c'_e. Otherwise (if no s-t path is left using only high-capacity edges), we know that the bottleneck edge has smaller capacity, and recursively search among the edges of smaller residual capacity. This way, in $O(\log m)$ iterations, we find the bottleneck edge, and each iteration takes time $O(m)$, to look for an s-t path with BFS. Hence, we can find the widest path in time $O(m \log m)$.

In fact, we can do even better than this: we can adapt Dijkstra's algorithm to find the widest s-t path with respect to c'_e (notice that the dynamic programming approach mentioned above would correspond to an

adaptation of the Bellman/Ford algorithm). For each node v, in our modification, we let r_v denote the width of the widest path from s to v discovered so far. When we consider nodes u for inclusion in the growing set S of nodes, we pick the endpoint of the edge $e = (v, u) \in S \times \overline{S}$ maximizing the quantity $\min(r(v), c_e)$, and include e and the endpoint u. The same inductive proof as for Dijkstra shows that this finds the widest s-t path, and using Fibonacci Heaps (:-), we can make it run in time $O(m + n \log n)$.

1 The Edmonds/Karp Algorithm

While having an actual polynomial-time algorithm is already progress, we would even prefer to have an algorithm whose running time does not depend on C at all. After all, our MST or shortest paths algorithms use edge costs c_e, but their running time depends only on the parameters m and n.[1] Such an algorithm is called *strongly polynomial*.

This is accomplished by the Edmonds/Karp algorithm, which gives yet another choice for the s-t path in the Ford/Fulkerson algorithm. If we look at the bad examples for Ford/Fulkerson, we see that the bad behavior can be attributed to two causes:

- We use paths with little capacity (addressed above).
- Our paths put flow on more edges than necessary.

The idea of the Edmonds/Karp algorithm is to attack the second weakness instead of the first, by always using a *shortest* s-t path in each iteration. Here, the length is counted as the number of edges of the path in the residual graph.

Notice that this length of the shortest path is always between 1 and $n - 1$. Furthermore, our intuition is that the shortest paths shouldn't really get shorter: our algorithm keeps adding flow, and *saturates* edges e (by making them reach $f_e = c_e$, or $f_e = 0$ for reverse edges) on the shortest paths early on. After that, we would expect that the shortest remaining paths keep getting longer, so after "not too many" iterations, the length should have reached n, at which point there is no s-t path remaining.

This intuition turns out to be correct, and forms the core of the proof of the following theorem:

Theorem 2 *The Edmonds/Karp algorithm terminates after $O(mn)$ iterations.*

Finding a shortest s-t path in the residual graph G_f is not very difficult. In fact, we do not even need to use Dijkstra's algorithm (or Bellman/Ford), as all edge lengths are only counted as 1. We can instead simply run BFS, starting at s, which takes time $O(m)$. Hence, the running time of the Edmonds/Karp algorithm is $O(m^2 n)$. Notice that, while this doesn't look too bad at first sight, if the graph is dense (i.e., it has $\Omega(n^2)$ edges), the running time is only bounded by $O(n^5)$, which is quite a lot.

Proof of Theorem 2. We write P_r for the shortest path used in iteration r of the Edmonds-Karp algorithm, and $|P_r|$ for the number of edges in it. Then, we know that the distance from s to t in iteration r was exactly $|P_r|$. Our proof will establish the following two key claims:

1. The distance from s to t never decreases.
2. Between two successive saturations of the same edge e (i.e., two times $r < r'$ at which f_e is *increased* to c_e, or decreased to 0), the distance from s to t strictly increases.

Let us first see that these two claims imply the Theorem. In each iteration, the Ford/Fulkerson algorithm saturates the bottleneck edge $\hat{e}$ on the path P_r. In particular, each iteration saturates some edge. After at most $m + 1$ iterations, the same edge must be saturated again, so every $m + 1$ iterations, the distance increases by at least 1. But it can only increase to at most n, so it increases at most n times. Hence, there can be at most $O(mn)$ iterations.

[1] Technically, of course, when they add b-bit numbers, they spend time $O(b)$, so all the arithmetic operations and comparisons actually do take time $\Theta(\log C)$. However, it is common to treat such arithmetic operations as taking only constant time, in which case we do want the algorithm to not depend on b.

So we want to prove the two claims, which correspond to our intuition as to why the Edmonds/Karp algorithm might work well. Let's think for a moment about the first claim. Assume for contradiction that the distance actually decreases from step r to $r+1$. Say the shortest path at time r is P, and the one at time $r+1$ is Q. Because Q is by assumption shorter than P, and P was the shortest path in the residual graph at time r, there must be at least one edge e in Q that is not in the residual graph at time r. The only reason e could appear at time $r+1$ is if P pushed flow on it, which Q can now push back the other way.

Suppose that e were actually the only such edge. (This is certainly not true, but we will take care of the more general case in a moment.) Then Q is of the form Q_0, followed by $\overleftarrow{e}$, followed by Q_1, where neither Q_0 noer Q_1 use edges from P in the opposite direction. On the other hand, P is of the form P_0, followed by e, followed by P_1. Because neither Q_0 nor Q_1 use edges in the opposite direction from P, they are also paths in the residual graph at time r, and therefore, both the path consisting of P_0 followed by Q_1, and the path consisting of Q_0 followed by P_1 are paths at time r. So they must both be at least as long as P, because P was shortest. This implies that $|Q_0| \geq |P_0| + 1$, and $|Q_1| \geq |P_1| + 1$, so Q is strictly longer than P.

We still have to take care of the case when there are multiple edges e used the opposite way. The generalization is proved in Lemma 3 below. Once we have proved that lemma, we want to verify that it implies both key claims above.

1. For the first part, we look at an arbitrary point in time r, and prove that the s-t distance from time r to $r+1$ does not increase. There are two cases: if the path chosen at time $r+1$ pushes flow the opposite direction from the one at r on any edge, then we can apply Lemma 3 to those two paths, and find that the s-t distance actually increased.

   Otherwise, the shortest s-t path P_{r+1} at time $r+1$ must actually be in the residual graph at time r, so it can be no shorter than the shortest path at time r. In both cases, the distance from s to t thus cannot decrease.

2. For the second part, we look at two successive saturations of the same edge e, happening at times r_1 and r_2. Then, there must be some time $\hat{r} \in (r_1, r_2)$ such that flow was pushed in the opposite direction at times r_1 and $\hat{r}$. Ideally, we would like to apply Lemma 3 to the times r_1 and $\hat{r}$, but we haven't made sure yet that the times also satisfy the second condition of the lemma.

   Instead, we let $r < r' \in (r_1, \hat{r})$ be such that (a) the paths at time r, r' push flow in opposite directions along at least one edge, and (b) $|r' - r|$ is minimized subject to constraint (a). We know that such r and r' exist, as r_1 and $\hat{r}$ are candidates; the only reason why we wouldn't chose them is because others are closer together. Now we want to claim that the times r, r' we constructed in this way, and the paths $P_r, P_{r'}$ along which flow is pushed at those times, satisfy the conditions of Lemma 3. The first condition simply follows by our requirement (a). The second one follows from the minimality of $|r' - r|$, for if there were a time $r'' \in (r, r')$ at which flow is pushed on an edge opposite to r or r', then r'' and r (or r'' and r') would form a pair closer in time.

   So we can apply Lemma 3, and find that the s-t distance at time r' is strictly greater than at time r. By part 1, it cannot decrease either from r_1 to r, or from r' to r_2, so it is strictly greater at time r_2 than at time r_1.

This completes the proof of the Theorem. ■

Lemma 3 *Let $r < r'$ be two iterations, and P, Q the shortest paths in those iterations, with the following properties:*

1. *There is at least one edge e such that P and Q push flow on e in opposite directions.*
2. *For all times $r'' \in (r, r')$, the path $P_{r''}$ chosen by the algorithm in iteration r'' does not push flow in the opposite direction from either P or Q, for any edge $e \in P_{r''}$.*

Then, $|P| < |Q|$.

Proof. We let the nodes of P be denoted by $1, 2, \ldots, k$, and the edges by $(1,2), (2,3), \ldots, (k-1,k)$. Also, we let $e_1 = (v_1 + 1, v_1), e_2 = (v_2 + 1, v_2), \ldots, e_\ell = (v_\ell + 1, v_\ell)$ be all the edges on which Q pushes flow in opposite direction from P. By the first assumption in the lemma, we know that $\ell \geq 1$. To get from $v_0 := s = 1$ to $v_1 + 1$, the path Q uses a subpath Q_0. Then, to get from v_j to $v_{j+1} + 1$, the path Q uses a subpath Q_j, and finally, to get from v_ℓ to k, it uses a subpath Q_ℓ. Because we assumed the e_j to be *all* edges that are used by Q in the opposite direction, and because no paths at time $t'' \in (r, r')$ used any edge from Q or P in the opposite direction, we know that each Q_j is also a path in the residual graph at time r.

Now, P is a shortest s-t path at time r, and so going from node a to b through $a+1, a+2, \ldots, b-1$ (as P does) must also be a shortest path from a to b. Otherwise, we could insert a shorter path, and obtain a shorter s-t path. In particular, we know that each Q_j, which is a path from v_j to $v_{j+1} + 1$ must contain at least $v_{j+1} + 1 - v_j$ edges, i.e., $|Q_j| \geq v_{j+1} + 1 - v_j$. (If $v_{j+1} < v_j$, then this is of course trivial.) Now, we know that the length of Q is $|Q| = \ell + \sum_{j=0}^{\ell} |Q_j|$. Substituting the inequality we argued a moment ago, this gives us that

$$|Q| \quad \geq \quad \ell + \textstyle\sum_{j=0}^{\ell-1}(v_{j+1} + 1 - v_j) + (k - v_\ell) \quad = \quad 2\ell + \sum_{j=0}^{\ell-1}(v_{j+1} - v_j) + (k - v_\ell) \quad = \quad 2\ell + k - 1.$$

In the last step, we used that the series telescopes — all terms except v_ℓ and $-v_0$ cancel out with the next terms. Now, because $\ell \geq 1$, we get that $|Q| \geq k + 1 > |P|$, which proves the lemma. ■