

Lecture 13: October 22

Lecturer: Micah Adler

Scribe: Jing Weng and Fang Yu

13.1 Network Flow

G_f is a residual network of flow f in graph $G = (V, E)$. We can think of an “augmenting path” p as a directed path from the source s to the sink t in the residual network G_f . Earlier we observed that we can increase the flow along every edge of the augmenting path by the bottleneck capacity of that path.

13.1.1 Ford-Fulkerson Algorithm with Edmonds-Karp Heuristics

- 1) $f = 0$
- 2) while there is an augmenting path p
- 3) find shortest augmenting path p
- 4) augment f by $b(p)$ units along p
- 5) return f

where $b(p)$ is the bottleneck capacity of the path p .

Theorem 13.1 *This algorithm finds maximum flow in time $O(|E|^2|V|)$*

Definition 13.2 $\delta_f(s, u)$ is the unweighted shortest path distance from s to u in the residual network G_f , where shortest path is defined in terms of the number of edges.

Lemma 13.3 $\delta_f(s, u)$ is non-decreasing as f changes. In other words, when we update the flow, the distance $\delta_f(s, u)$ can only increase.

Definition 13.4 Edge (u, v) is **critical** for flow f if (u, v) is on p and $c(u, v)$ in G_f is bottleneck capacity $b(p)$, where p is the augmenting path chosen from G_f .

First suppose lemma 13.3 is right, to prove theorem 13.1 we need to show:

for any (u, v) between two occasions where (u, v) is critical, $\delta_f(s, u)$ increases by at least 2.

then, we can get

Total Augmentations: $|V||E|$

One edge: maximum path length / 2 = $|V|/2$

Total pairs (u, v) : $2|E|$

Proof: Let (u, v) be critical in f . We say that an edge (u, v) in a residual network G_f is critical on an augmenting path p if the residual capacity of p is the residual capacity of (u, v) . After we have augmented flow along an augmenting path, any critical edge on the path disappears from the residual network. At least one edge on any augmenting path must be critical.

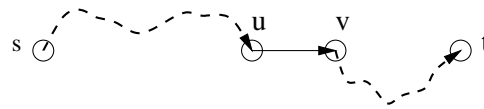


Figure 13.3: G_f . After the augmentation the edge (u, v) goes away. For it to be critical again, it must reappear in the graph. This happens when flow is pushed in the opposite direction along this edge.

$$\delta_f(s, u) = \delta_f(s, v) - 1 \tag{13.1}$$

- f'' : next flow where (u, v) is in residual graph
- f' : flow right before f''

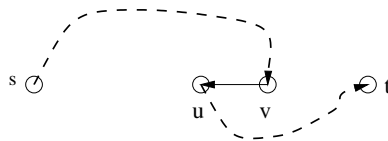


Figure 13.4: $G_{f'}$.

Since (u, v) is in f'' and not in f' the augmenting path must contain (v, u) . Thus

$$\delta_{f'}(s, v) = \delta_{f'}(s, u) - 1 \tag{13.2}$$

Combining:

$$\delta_{f'}(s, v) = \delta_{f'}(s, u) - 1 \geq \delta_f(s, u) - 1 = \delta_f(s, v) - 2 \tag{13.3}$$

■

now we prove Lemma 13.3

Proof: Assume that we have an augmenting path from f to f' and a vertex v , such that the shortest path distance decreases:

$$\delta_{f'}(s, v) < \delta_f(s, v) \tag{*}$$

- If there are more than one such vertices, let v be the closest one to s in $G_{f'}$ for which $(*)$ holds. The distance $\delta_{f'}(s, v)$ is the smallest for this vertex.
- Let u be the second to last vertex on the shortest path from s to v in $G_{f'}$.

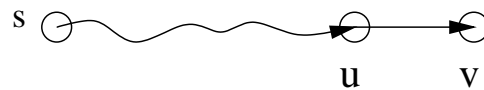


Figure 13.1: $G_{f'}$

Claim 13.5 $(u, v) \notin E_f$, i.e., the edge (u, v) is not an edge in the residual network for the flow f .

Proof: Otherwise, $\delta_f(s, v) \leq \delta_f(s, u) + 1 \leq \delta_{f'}(s, u) + 1 = \delta_{f'}(s, v) \implies$ contradiction ■

In the residual network for f' the edge (u, v) is present - it's on the shortest path from s to v . But we also know that this edge is not present in the residual network for f . That means that when we perform the augmentation from f to f' , somehow we have to cause that edge to appear. The only way that we can do that is by decreasing the amount of flow that is going from u to v . That can only happen if we have an augmenting path, that travels in the other direction (from u to v):

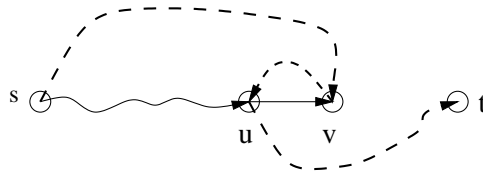


Figure 13.2: $G_{f'}$, the residual network for flow f' . Since we have an augmentation from f to f' , that means we have an augmenting path from s to t (shown as a dashed line), which must include an edge from v to u to cause the edge (u, v) to appear in $G_{f'}$.

Thus the augmenting path that takes us from f to f' has to contain the edge (v, u) . In other words, this edge has to be on the shortest path from s to t . The node v is then on the shortest path from s to u and it is the predecessor to u on that shortest path.

Therefore, we can write that:

$$\delta_f(s, u) = \delta_f(s, v) + 1 \tag{13.4}$$

We can re-write it as

$$\delta_f(s, v) = \delta_f(s, u) - 1 \tag{13.5}$$

The term $\delta_f(s, u) - 1$ has to be less than or equal to the term $\delta_{f'}(s, u) - 1$ because u is the closest to v in the residual network $G_{f'}$.

$$\delta_f(s, v) = \delta_f(s, u) - 1 \leq \delta_{f'}(s, u) - 1 \tag{13.6}$$

Since we have defined u to be the closest vertex to v we have

$$\delta_{f'}(s, u) - 1 = \delta_{f'}(s, v) - 2 \tag{13.7}$$

which gives us

$$\delta_f(s, v) = \delta_f(s, u) - 1 \leq \delta_{f'}(s, u) - 1 = \delta_{f'}(s, v) - 2 \tag{13.8}$$

collapsing inequality (13.8) we get

$$\delta_f(s, v) \leq \delta_{f'}(s, v) - 2, \tag{13.9}$$

Which contradicts our assumption (*), therefore the lemma must be true.

The distance of u from the source is initially at least 1, and until it becomes unreachable from the source, if ever, its distance is at most $|V| - 2$. Thus, (u, v) can become critical at most $O(|V|)$ times. Since there are $O(|E|)$ pairs of vertices that can have an edge between them in a residual graph, the total number of critical edges during the entire execution of the algorithm is $O(|V||E|)$. Each augmenting path has at least one critical edge, therefore the total number of flow augmentation performed by the algorithm is at most $O(|V||E|)$.

Since each iteration of the algorithm can be implemented in $O(|E|)$ time when augmenting path is found by breadth-first search, the total running time is $O(|V||E|^2)$.

That completes the proof of the Theorem. ■

The fastest algorithm so far known [K74] has the running time $O(|V|^3)$. Algorithm of Goldberg-Tarjan [GT88] has the running time $O(|V||E|\log \frac{|V|^2}{|E|})$. Both of these algorithms work on sparse graphs. For simple networks the algorithm of Even-Tarjan [ET75] gives the running time of $O(|E|\sqrt{|V|})$. More recent and faster algorithms are based on randomization (Karger, MIT).

13.1.2 Application: Bipartite Matching

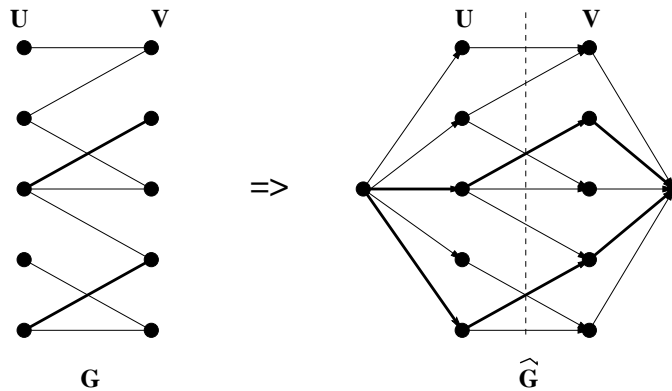


Figure 13.5: G is a bipartite graph, with thick edges denoting a matching from U to V . \hat{G} is a corresponding flow network, in which thick edges denote a flow, corresponding to the matching in G

Given a bipartite graph G , let \hat{G} be the corresponding flow network. To construct \hat{G} we add two vertices to G : source s and sink t . Then we add edges from s to every vertex in U , and edges from every vertex in V to t . The capacity of every edge is set to 1.

Claim 13.6 G has a maximum matching of size $k \Leftrightarrow \hat{G}$ has a maximum flow of size k .

Proof: In one direction (\Rightarrow): Route one unit of flow on each edge of the matching. Send one unit of flow from s to each edge incident to a node in the matching.

In the other direction (\Leftarrow): Flow must be integral, and capacity across an $s - t$ cut $c(\{s\} \cup u, v \cup \{t\}) = k$

If flow from s to t is k , we know that we have k edges, since the capacity of each edge is 1. Therefore, we have a matching of size k . ■

13.2 Randomized Algorithms

This section introduces the notion of a randomized algorithm and presents a randomized version of **Quick-sort**.

13.2.1 Quicksort

Quicksort is a sorting algorithm that is based on the divide and conquer paradigm. To sort an array of elements, we pick one element to be the **pivot**. We then split the original array into two smaller arrays, one containing elements that are less than the pivot, and the other containing elements that are greater than the pivot. We then sort those two arrays recursively, and recombine the results.

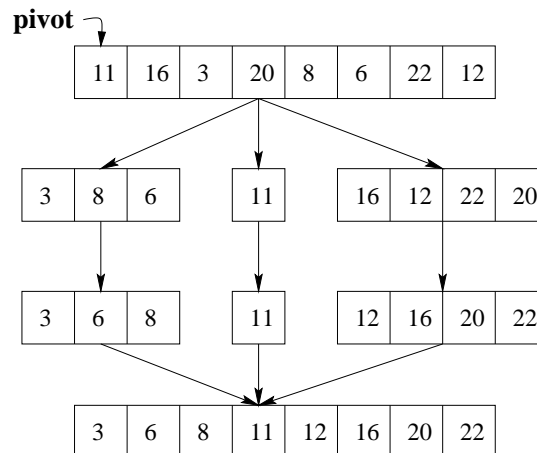


Figure 13.6: An example of Quicksort

The running time of this algorithm depends on how we choose the pivot element. One approach to do that is to pick it randomly.

Definition 13.7 *Average run time:* $\bar{T}(n) = \max_{|x|=n} E[T(x)]$, where E is the expected value with respect to random choices made by the algorithm.

Theorem 13.8 *For Quicksort, $\bar{T}(n) = \theta(n \log n)$. Specifically, for $n \geq 2$, $\bar{T}(n) \leq a \ln n$, where a is some constant.*

References

- 1 S.KAVALENKA and D. LISIN Scribe Notes for Lecture12, UMASS CMPSCI 611 *Advanced Algorithms, Fall2001*.

- CLR90 T. CORMEN, C. LEISERSON, and R. RIVEST, *Introduction to Algorithms*, McGraw-Hill, 1990.
- ET75 S. EVEN and R.E. TARJAN, Network flow and testing graph connectivity, *SIAM Journal on Computing* **4**, 1975, pp. 507–518.
- GT88 A.V. GOLDBERG and R.E. TARJAN, A new approach to the maximum flow problem, *Journal of the ACM* **35**, 1988, 921–940.
- K74 A.V. KARZANOV, Determining the maximal flow in a network by the method of preflows, *Soviet Math. Dokl.* **15**, 1974, 434–437.