MIT 6.854/18.415: Advanced Algorithms Spring 2016

Lecture 8 – February 29, 2016

Prof. Ankur Moitra Scribe: Sidd Seethepalli, Rajeev Parvathala, Shraman Ray Chaudhuri

1 Last Time

Last time we talked about flow decomposition and Ford-Fulkerson, and we proved that the max flow equals the min cut.

When applying Ford-Fulkerson last time, we faced the problem of how to pick the augmenting path at every step. Given adversarial path selection, we found that on real valued capacity graphs, the algorithm could run forever without terminating. Worse, the flow value might even converge to a value less than the actual max flow. On integer capacity graphs with m edges and values bounded by U, the algorithm could take $\Theta(mU)$ iterations.

This is pseudopolynomial in the input, since it is exponential in the number of bits $\lg U$ and polynomial in the size of m. Note that for graphs with integer valued capacities, the algorithm always outputs an integer valued max flow. We will leverage this fact in our development of a better integer max-flow algorithm with a Capacity Scaling technique.

2 Capacity Scaling

The idea behind capacity scaling is to augment the highest flow paths first to reach our maximum flow faster. We do this by limiting our scope to a **D-residual graph**, i.e. a residual graph with all edges with $< D$ capacity removed.

2.1 Framework

Our capacity scaling algorithm works as follows:

```
Set D = U
While D ≥ 1:
    Let H_f(D) be the D-residual graph
    While ∃ s − t path w in H_f(D):
        Augment along w by lowest edge capacity (i.e. Ford-Fulkerson step)
        Update D-residual
    Set D = D/2
```

2.2 Analysis

Let $m = |E|$, $n = |V|$, f^* be a maximum flow, and f be the current flow produced by a given step in the algorithm. We now prove the following theorem:

Theorem [Edmonds-Karp, Dinitz] *Capacity scaling* (1) *computes a maximum flow and* (2) *can be implemented in* $O(m^2(1 + \lg |U|))$ *time.*

To prove (1), we have the following lemma:

Lemma 1. *When a capacity scaling algorithm terminates, it produces a max flow* f^**.*

Proof. When running Ford-Fulkerson any regular residual graph H, we are guaranteed to achieve a max flow f^* (proven in last lecture). When $D = 1$, $H_f(D) = H$, i.e. our inner loop in the capacity scaling framework becomes an instance of the Ford-Fulkerson algorithm running on a regular residual. Therefore, at termination, we are guaranteed to achieve f^*.

Finally, to prove (2), we have:

Lemma 2. *When the outer loop terminates for some* D*,*

$$|f^*| \leq |f| + Dm$$

Proof. We know there is an $s - t$ in residual H of value $|f^*| - |f|$. Let S be the nodes reachable from s in $H_f(D)$. Clearly, the capacity of the cut between S and its complement $\leq Dm$. Since max flow $\leq$ any cut, we're done.

Finally, each augmentation must increase the flow value by at least D, since it's along a path in the D-residual graph. Thus there can be at most $O(m)$ augmentations per iteration of the outer loop.

2.2.1 Remarks

The running time of $O(m^2(1+\lg U))$ is indeed polynomial in the number of bits needed to describe our problem (i.e. "description length")—however, this is only **weakly polynomial**.

A **strongly polynomial** algorithm depends only on the *number* of values used to describe the problem (i.e. m and n) and is independent of the *size* the values can take on.

There exist many strongly polynomial time algorithms for the max flow problem. One simple algorithm is to use BFS to find the shortest augmenting path on each iteration of Ford-Fulkersons, resulting in an $O(m^2n)$ running time. Consult the 6.854 Notes from Fall 2013 [Karger] for the formal proof.

3 Min-Cost Flows

In the problem of Min-Cost Flow, each edge has both a capacity and a real number cost, c, associated with it. The goal is now to find the max $s - t$ flow that minimizes $\sum_{(a,b)\in E} f(a, b)c(a, b)$.

3.1 A Simpler Problem: Min-Cost Bipartite Matching

Just for review, a **matching** $M \in E$ is a set of edges such that each vertex in the graph G is incident on at most one edge $e \in M$.

The **Bipartite Perfect Matching** problem is as follows: We are given a bipartite graph $G = (V, E)$, where $V = A \cup B$, $|A| = |B| = n$, and $E \subset A \times B$. We want to find out whether or not $\exists$ a perfect matching M such that M covers all the vertices in the graph. Just to be clear, $\forall e \in E, e = (a, b)$ for $a \in A, b \in B$, and the edges are undirected.

We can construct a reduction to a max flow problem. Create a new vertex s, and $\forall a_i \in A$, construct an edge (s, a_i) in a new graph G'. Then, $\forall e = (a_i, b_i) \in E$, construct a directed edge (a_i, b_i) in G'. Then, construct another new vertex t, and add new edges $(b_i, t), \forall b_i \in B$. Let each of the edges in our new graph have capacity equal to 1. Remember that $|A| = |B| = n$. So in our new graph G', if $\exists$ flow $f, |f| = n$, then there exists a perfect bipartite matching in our graph; the contrapositive of this statement is therefore also true.

3.2 Back to Min-Cost Flow

So how does our Min-Cost problem relate to the Bipartite Perfect Matching problem? Essentially, in the case of a bipartite graph, we want to find the perfect matching with the minimum possible cost.

Let us present the following algorithm: Given some perfect matching M, construct the residual graph H_M according to the following rules:

- If $(a, b) \notin M$: add the directed edge (a, b) to H_M, with $c_h(a, b) = c(a, b)$.
- If $(a, b) \in M$: add the directed edge (b, a) to H_M, with $c_h(b, a) = -c(a, b)$.

We now present the following Lemma:

Lemma. *Let M be a perfect matching. Then the constructed residual H_M has a cycle of negative total cost iff M is not a min cost perfect matching.*

Proof. Let C be a negative cost cycle in H_M: $c(C) < 0$. Then, augmenting along C in H_M will give us a new perfect matching M', according to the following rules:

To construct M', begin with M. Then, for each edge $e = (a_i, b_i) \in H_M$ that is included in C, add the undirected e to M'. And for each edge $e = (b_i, a_i) \in H_M$ that is in the cycle, remove (a_i, b_i) from M'. Once you have done this for all edges in the cycle, you will be left with a new perfect matching.

Additionally, the new cost $c(M') = c(M) + c(C) < c(M)$. Let M' be any lower cost perfect matching. Then we will define the symmetric difference $M \Delta M' = \{e \in E : e \in M \oplus e \in M'\}$; basically, these are the edges that are in either M or M', but not both.

Note that $\sum_{i=1}^{k} c_{H_M}(C_i) = c(M') - c(M) < 0$. Therefore, the cycle containing precisely these edges has a negative total cost, and we are done.

3.3 Klein's Cycle Cancelling Algorithm

Find a max $s-t$ flow in G;
while $\exists$ *a negative cost cycle* C **do**
  augment along the cycle by $\Delta_{H_M}(C)$;
end

The question of how to choose the cycle remains, however.
Choosing any negative cost cycle via Bellman-Ford results in pseudopolynomial runtime.
Choosing the most negative cycle is NP-Hard!

3.3.1 Next Time

We will see how to choose cycles to execute Klein's Cycle Cancelling Algorithm efficiently for a perfect matching, then look at the general case.