COMS 4995-3: Advanced Algorithms Feb 27, 2017

Lecture 11 – The Maximum Flow Problem

Instructor: *Alex Andoni* Scribes: *Chengyu Lin*

1 Introduction

Last lecture we introduced the notion of $s-t$ *flows in directed graphs*, $s-t$ *cut* and the decomposition theorem where every flow can be decomposed into at most m *path flows* and at most m *cycle flows*. Also we showed that for every $s-t$ flow f and every $s-t$ cut $(S, \overline{S})$, the value of flow f is at most the capacity of the cut: $|f| \leq u(S)$.

Today we continue discussion on the Maximum Flow Problem – to find the maximum $s-t$ flow on a graph. We'll show that the procedure that we gave last lecture – by keep finding a non-empty $s-t$ path flow – doesn't give the maximum flow. Then we'll correct it by introducing the notion of *Residual Graph* and *augmenting path*. We'll present and analyze the *Ford-Fulkerson algorithm* which solves the max-flow problem, and discuss how to find a way to improve it's performance.

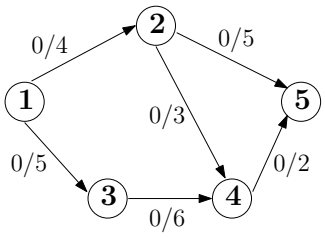
2 Procedure of Finding Path Flows

We start by reviewing the procedure that we gave last lecture:

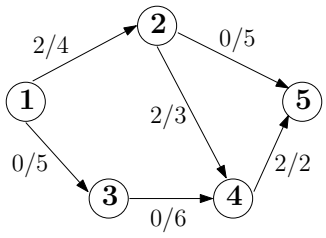
1. Find an arbitrary $s-t$ path flow P.
2. Set $f_{e\in P}(e) = \min_{e'\in P} u(e')$.
3. Repeat: find a new path P' where, for all constituent edge e, $u(e) - f(e) > 0$
   (a) Let $\delta_p = \min_{e'\in P'} u(e') - f(e')$
   (b) $f^{new}(e) = f^{old}(e) + \delta_p$, $\forall e \in P'$
   (c) Set $f^{old} = f^{new}$

Note that this procedure does not always give us the max-flow of a graph. Consider the following example as shown in figure 1. Suppose we want to find the max-flow between vertex 1 and 5. Starting with a path flow $1 \to 2 \to 4 \to 5$, we can see this procedure terminates after finding another path flow $1 \to 2 \to 5$. So we can obtain a flow of value 4. But clearly if we can push the flows from vertex 2 to vertex 4 back, and redirect it to vertex 5, we can find a new path flow and increase the total value of the flow.

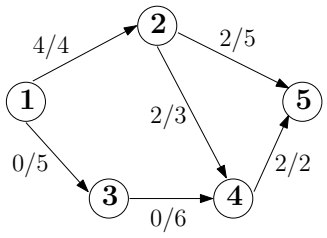
Later we show how to fix this problem by allowing redirecting some flows.



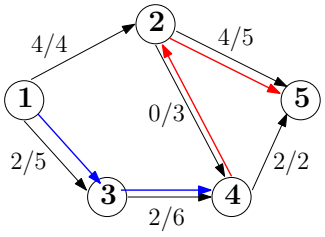
(a) Graph G



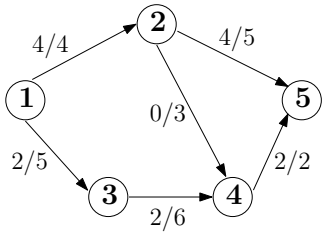
(b) Starting with a path flow $1 \to 2 \to 4 \to 5$



(c) Find a path flow $1 \to 2 \to 5$, the procedure terminates here



(d) Redirect flow $2 \to 4$ back to 5, make up a new flow $1 \to 3 \to 4$



(e) A max-flow of graph G

Figure 1: Bad example for the procedure

3 Ford-Fulkerson algorithm

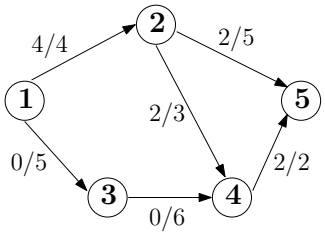
We begin with 2 important definitions.

Definition 1. *For graph $G = (V, E)$ and flow f on G, a Residue graph G_f is a graph on the same set of vertices where*

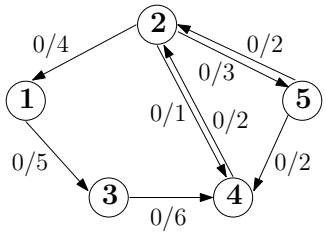
$$u_f(v,w) = \begin{cases} u(v,w) - f(v,w) & \text{if } (v,w) \in E \quad \textit{(forward)} \\ f(w,v) & \text{if } (w,v) \in E \quad \textit{(backward)} \end{cases}$$

Definition 2. *An augmenting path P is a $s-t$ path in residue graph s.t. $\delta_f(P) = \min_{e \in P} u_f(e) > 0$.*

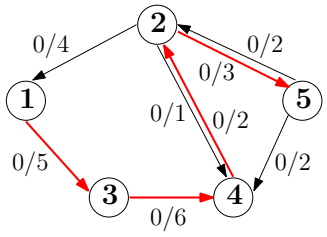
Note that for simplicity we assume that for any pair of vertices v and w, only one of (v, w) or (w, v) appears in E.



(a) Graph G with flow f



(b) Residue graph G_f



(c) A augmenting path with $\delta_P = 2$

Figure 2: Residue graph and augmenting path

Then we present the Ford-Fulkerson algorithm based on augmenting path:

Initialize: set $f(v,w) = 0$ for every $(v,w) \in E$;
$G_f = G$;
while *there exists an augmenting* $s-t$ *path* P *in* G_f **do**
  $\delta = \min_{e \in P} u_f(e)$;
  foreach $(v,w) \in P$ **do**
    if (v,w) *is a forward edge* **then**
      $f(v,w) = f(v,w) + \delta$;
    else
      $f(w,v) = f(w,v) - \delta$;
    end
  end
  update G_f;
end

It's easy to check that, after each step of augmenting, what we obtain is still a valid flow.

Here's the first question, how fast does it converge? Is it possible that the procedure of finding augmenting path will loop forever? Assume that all edge capacities are bounded integers $1, 2, \ldots, U$. We show that Ford-Fulkerson algorithm terminates in a reasonable time.

Theorem 3. *Ford-Fulkerson algorithm runs in* $O(nmU)$ *time.*

Proof. First we have a natural upper bound for max-flow $|f| \leq n \cdot U$. Because there are at most n edges going out of s.

Each time we push an augmenting path P, by definition $\delta_f(P) > 0$ so $\delta_f(P) \geq 1$. We'll increase the flow value by at least 1. Hence there'll be at most $n \cdot U$ iterations.

For each iteration, we can find an augmenting path by BFS, which takes $O(m)$ time.

To sum up, we have the running time of Ford-Fulkerson algorithm is $O(nmU)$. □

As for the correctness of the algorithm, we show a stronger theorem.

Before we showed for any $s-t$ flow and any $s-t$ cut, the value of flow cannot exceed the capacity of cut. Here we show a tight relation between them.

Theorem 4 (Max-flow min-cut theorem)**.**

$$\max_{\text{flow } f} |f| = \min_{s-t \text{ cut } (S,\bar{S})} u(S)$$

Proof. Let f be the output of Ford-Fulkerson algorithm, and the residue graph G_f. Let S be the set of vertices reachable from s in G_f.

First $t \notin S$, otherwise we can find an augmenting path from s to t in G_f and increase the value of the flow. Therefore $(S, \bar{S})$ is a $s-t$ cut.

Then for all edges (v,w) where $v \in S$ and $w \in \bar{S}$, by definition we have (v,w) is saturated $f(v,w) = u(v,w)$.

Now we decompose max-flow f into paths and cycles. Each path has to cross the cut $(S, \bar{S})$ exactly once. If more than 1 crossings, there must be an edge (w,v) where $w \in \bar{S}$ and $v \in S$ on this path that has positive flows, and there should be a backward edge (v,w) with positive capacity in the residue graph

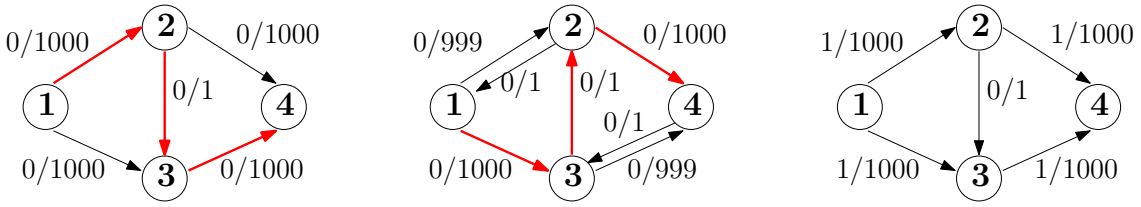
G_f. So v and w should be both reachable from s in G_f. But by definition of S, v is reachable from s in G_f but w isn't.

As each path in the decomposition has to cross cut $(S, \bar{S})$ exactly once, the value of flow should be no less than the capacity of cut $|f| \geq u(S)$. Combine with previous result where the value of any flow cannot exceed the capacity of any cut, we have $|f| = u(S)$. And f is a max-flow and the defined $(S, \bar{S})$ is a min-cut. □

4 Boosting Ford-Fulkerson Algorithm

Consider the running time of Ford-Fulkerson algorithm $O(nmU)$. Because of its dependence of U and the length of input description is $O(m \log U)$, it's a pseudopolynomial. We expect the running time of our algorithm to be weakly polynomial $poly(n, m, \log U)$ or even strongly polynomial $poly(n, m)$.

But the performance of Ford-Fulkerson algorithm may not meet our needs. Consider the following bad example in figure 3. Each iteration we can only increase the value of flow by 1. To obtain the max-flow we need $2U = 2000$ iterations.



(a) Original graph and an augmenting path

(b) Residue graph, with another augmenting path

(c) After 2 iterations, we only get a flow of value 2

Figure 3: Bad example for Ford-Fulkerson algorithm

Look at the bad example, if we have a clever way of finding the augmenting path, we may have a better running time. Here are some possible solutions:

- Find the shortest augmenting path at each iteration
- Find the augmenting path with higher flow
- Choose one augmenting path randomly
- Build the flow iteratively. We may do better if we already have some guarantees on the partial flow.

Here we'll try the second approach, at each iteration we try to find the augmenting path with the highest possible flow.

Definition 5. *For augmenting path P in residue graph G_f, let the bottleneck capacity of P be*

$$u_f(P) \triangleq \min_{e \in P} u_f(e)$$

And our new algorithm is exactly the same as Ford-Fulkerson algorithm, except when finding an augmenting path, we find one which maximizes $u_f(P)$.

The correctness follows directly from Ford-Fulkerson algorithm. As for how to find P which maximizes $u_f(P)$, we can use dynamic programming which is similar to Dijkstra algorithm. But here we'll do a binary search:

- Suppose we are working on residue graph G_f. Initially set $l = 1$ and $r = U$.
- Let $v = (l + r)/2$.
- Take all edges of value $\geq v$, delete the rest.
- If there exists a $s - t$ path, then $u_f(P) \geq v$, we recurse on $[v, r]$. Otherwise we recurse on $[l, v]$.

Using the procedure above, it takes $\log U$ iterations for binary search. For each iteration, if we use BFS it takes $O(m)$ time. So in total we need $O(m \log U)$ time to find an augmenting path.

Next to bound the number of augmenting path. We'll give the full proof in the next lecture. The main idea is, let g be the max-flow for residue graph G_f. It can be decomposed into at most m path flows. So there exists a path of value $\geq |g|/m$. Which means after we find an augmenting path with the maximum bottleneck capacity, the value of max residue flow drops by a factor of $\leq (1 - \frac{1}{m})$. Therefore after $k = O(m \log(nU))$ iterations we can obtain the max-flow.