Advanced Graph Algorithms and Optimization Spring 2020

Classical Algorithms for Maximum Flow II

Rasmus Kyng, Scribe: Timon Knigge *Lecture 11 — Wednesday, May 6th*

1 Overview

In this lecture we continue looking at classical max flow algorithms. We derive Dinic's algorithm which (unlike Ford-Fulkerson) converges in a polynomial number of iterations. We will also show faster convergence on unit capacity graphs. The setting is the same as last lecture: we have a directed graph $G = (V, E, \boldsymbol{c})$ with positive capacities on the edges. Our goal is to solve the maximum flow problem on this graph for a given source s and sink $t \neq s$:

$$\max_{F>0} F \qquad \text{s.t.} \quad \boldsymbol{B}\boldsymbol{f} = F\boldsymbol{b}_{s,t} \qquad \boldsymbol{0} \leq \boldsymbol{f} \leq \boldsymbol{c} \tag{1}$$

2 Blocking flows

Let $\boldsymbol{f}$ be any feasible flow in G and let $G_{\boldsymbol{f}}$ be its residual graph. Observe that we can partition the vertices of $G_{\boldsymbol{f}}$ into 'layers': first s itself, then all vertices reachable from s in one hop, then all vertices reachable in two hops, etc. For each vertex $v \in V$ define its **level** in $G_{\boldsymbol{f}}$ as the length of the shortest path in $G_{\boldsymbol{f}}$ from s to v, denoted by $\ell_{G_{\boldsymbol{f}}}(v)$ (or just $\ell(v)$ if the graph is clear from context). An edge $(u, v) \in E$ can only take you up 'one level': if $\ell(v) \geq 2 + \ell(u)$ this would imply we can find a shorter s-v path by appending (u, v) to the shortest s-u path. However, edges can 'drop down' multiple levels (or be contained in the same level).

A key strategy in Dinic's algorithm will be to focus on 'short' augmenting paths. We can use the levels we defined above to isolate a subgraph of $G_{\boldsymbol{f}}$ containing all information we need to find shortest paths:

Definition 2.1. Call an edge (u, v) **admissible** if $\ell(u) + 1 = \ell(v)$. Let L be the set of admissible edges in $G_{\boldsymbol{f}}$ and define the **level graph** of $G_{\boldsymbol{f}}$ to be the subgraph induced by these edges.

Definition 2.2. Define a **blocking flow** in a residual graph $G_{\boldsymbol{f}}$ to be a flow $\hat{\boldsymbol{f}}$ feasible in $G_{\boldsymbol{f}}$, such that $\hat{\boldsymbol{f}}$ only uses admissible edges. Furthermore we require that for any s-t path in the level graph of $G_{\boldsymbol{f}}$, $\hat{\boldsymbol{f}}$ saturates at least one edge on this path.

The last condition makes a blocking flow 'blocking': it blocks any shortest augmenting path in $G_{\boldsymbol{f}}$ by saturating one of its edges. Note however that a blocking flow is not necessarily a maximum flow in the level graph:

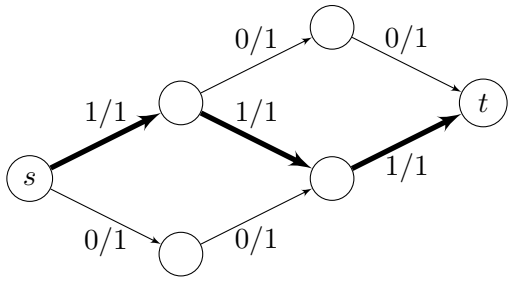

Figure 1: A blocking flow is not a maximum flow.

3 Dinic's algorithm

We can now formulate Dinic[1]'s algorithm: start with $\boldsymbol{f} = \mathbf{0}$, and then repeatedly add to $\boldsymbol{f}$ a blocking flow $\hat{\boldsymbol{f}}$ in $G_{\boldsymbol{f}}$, until no more s-t paths exist in $G_{\boldsymbol{f}}$.

Note that by doing a path decomposition on $\hat{\boldsymbol{f}}$ and adding these paths to $\boldsymbol{f}$ one by one, we see that our algorithm is a 'special case' of Ford-Fulkerson (with a particular augmenting path strategy) and hence inherits all the behaviors/bounds we proved in the last lecture.

Lemma 3.1. *Let $\boldsymbol{f}$ be a feasible flow, $\hat{\boldsymbol{f}}$ a blocking flow in $G_{\boldsymbol{f}}$ and define $\boldsymbol{f}' = \boldsymbol{f} + \hat{\boldsymbol{f}}$ (think of $\boldsymbol{f}$ and $\boldsymbol{f}'$ to be the flows at some steps k and $k+1$ in the algorithm). Then $\ell_{G_{\boldsymbol{f}'}}(t) \geq \ell_{G_{\boldsymbol{f}}}(t) + 1$.*

Proof. Let L, L' be the edge sets of the level graphs of $G_{\boldsymbol{f}}, G_{\boldsymbol{f}'}$ respectively. We would now like to show that $d_{L'}(s,t) \geq 1 + d_L(s,t)$. Let's first assume that in fact $d_{L'}(s,t) < d_L(s,t)$. Now take a shortest s-t path in the level graph of $G_{\boldsymbol{f}'}$, say $s = v_0, v_1, \ldots, v_{d_{L'}(s,t)} = t$. Let v_j be the first vertex along the path such that $d_{L'}(s,v_j) < d_L(s,v_j)$. As $d_{L'}(s,t) < d_L(s,t)$, such a v_j must exist.

We'd like to understand the edges in the level graph L'. Let $E(G_{\boldsymbol{f}})$ denote the edges of the $G_{\boldsymbol{f}}$, and let $\mathrm{rev}(L)$ denote the set of reversed edges of L. The level graph edges of $G_{\boldsymbol{f}'}$ must satisfy $L' \subseteq L \cup \mathrm{rev}(L) \cup (E(G_{\boldsymbol{f}}) \setminus L)$.

In our s-t path in L', the vertex v_j is reached by an edge from v_{j-1}, and the level of v_{j-1} in $G_{\boldsymbol{f}}$ and $G_{\boldsymbol{f}'}$ are the same, so the edge $(v_{j-1}, v_j) \in L$ skips at least one level forward in $G_{\boldsymbol{f}}$. But, edges in L do not skip a level in $G_{\boldsymbol{f}}$, and edges in $\mathrm{rev}(L)$ or $E(G_{\boldsymbol{f}}) \setminus L$ do not move from a lower level to higher level in $G_{\boldsymbol{f}}$. So this edge cannot exist and we have reached a contradiction.

Now suppose that $d_{L'}(s,t) = d_L(s,t)$. This means there is an s-t path in L' using edges in L – but such a path must contain an edge saturated by the blocking flow $\hat{\boldsymbol{f}}$.

(Note: the reason this path must use only edges in L is similar to the previous case: the other possible types of edges do not move to higher levels in $G_{\boldsymbol{f}}$ at all, making the path too long if we use any of them.)

So we must have $d_{L'}(s,t) \geq 1 + d_L(s,t)$. □

An immediate corollary of this lemma is the convergence of Dinic's algorithm:

Theorem 3.2. *Dinic's algorithm terminates in $O(n)$ iterations.*

[1]Sometimes also transliterated as Dinitz.

Proof. By the previous lemma, the level of t increases by at least one each iteration. A shortest path can only contain each vertex once (otherwise it would contain a cycle) so the level of any vertex is never more than n. □

For graphs with unit capacities ($c = 1$) we can prove even better bounds on the number of iterations.

Theorem 3.3. *On unit capacity graphs, Dinic's algorithm terminates in*

$$O\left(\min\{m^{1/2}, n^{2/3}\}\right)$$

iterations.

Proof. We prove the two bounds separately.

1. Suppose we run Dinic's algorithm for k iterations, obtaining a flow $\boldsymbol{f}$ that is feasible but not necessarily yet optimal. Let $\tilde{\boldsymbol{f}}$ be an optimal flow in $G_{\boldsymbol{f}}$ (i.e. $\boldsymbol{f} + \tilde{\boldsymbol{f}}$ is a maximum flow for the whole graph), and consider a path decomposition of $\tilde{\boldsymbol{f}}$. Since after k iterations any s-t path has length k or more, we use up a total capacity of at least $\text{val}(\tilde{\boldsymbol{f}})k$ across all edges. But the edges in $G_{\boldsymbol{f}}$ are either edges from the original graph G or their reversals (but never both) meaning the total capacity of $G_{\boldsymbol{f}}$ is at most m, hence $\tilde{\boldsymbol{f}} \leq m/k$.

   Recalling our earlier observation that our algorithm is a special case of Ford-Fulkerson, this implies that our algorithm will terminate after at most another m/k iterations. Hence the number of iterations is bounded by $k + m/k$ for any $k > 0$. Substituting $k = \sqrt{m}$ gives the first desired bound.

2. Suppose again that we run Dinic's algorithm for k iterations obtaining a flow $\boldsymbol{f}$. The level graph of $G_{\boldsymbol{f}}$ partitions the vertices into sets $D_i = \{u \mid \ell(u) = i\}$ for $i \geq 0$. As shown before, the sink t must be at a level at least k, meaning we have at least this many non-empty levels starting from $D_0 = \{s\}$. To simplify, discard all vertices and levels beyond $D_{\ell(t)}$.

   Now, consider choosing a level I uniformly at random from $\{1, \ldots, k\}$. Since there are at most $n - 1$ vertices in total across these levels, $\mathbb{E}[|D_I|] \leq (n-1)/k$, and by Markov's inequality

   $$\Pr[|D_I| \geq 2n/k] \leq (n-1)/n < 1/2.$$

   Thus strictly more than $k/2$ of the levels $i \in \{1, \ldots, k\}$ have $|D_i| < 2n/k$ and so there must two adjacent levels j, $j+1$ for which this upper bound on the size holds. There can be at most $|D_j| \cdot |D_{j+1}| \leq 4n^2/k^2$ between these levels, and by the min-cost max-flow theorem we saw in the previous lecture, this is an upperbound on the flow in $G_{\boldsymbol{f}}$, and hence on the number of iterations still needed for our algorithm to terminate.

   This means the number of iterations is bounded by $k + 4n^2/k^2$ for any $k > 0$, which is $O(n^{2/3})$ at $k = 2n^{2/3}$.

□

4 Finding blocking flows

What has been missing from our discussion so far is the process of actually finding a blocking flow. We can achieve this using repeated depth-first search. We repeatedly do a search in the level graph (so only using edges L) for s-t paths and augment these. We erase edges whose subtrees have been exhausted and do not contain any augmenting paths to t. Pseudocode for the algorithm is given below.

Algorithm 1: FindBlockingFlow

```
f ← 0;
H ← L;
repeat
    P ← Dfs(s, H, t);
    if P ≠ ∅ then
        Let f̂ be a flow that saturates path P.
        f ← f + f̂;
        Remove from H all edges saturated by f̂.
    else
        return f;
    end
```

Algorithm 2: Dfs(u, H, t)

```
if u = t then
    return the path P on the dfs-stack.
end
for (u, v) ∈ H do
    P ← Dfs(v, H, t);
    if P ≠ ∅ then
        return P;
    else
        Erase (u, v) from H.
    end
end
return ∅;
```

Lemma 4.1. *For general graphs,* FindBlockingFlow *returns a blocking flow in $O(nm)$ time. Hence Dinic's algorithm runs in $O(n^2m)$ time on general capacity graphs.*

Proof. First, consider the amount of work spent pushing edges onto the stack which eventually results in augmentation along an s-t consisting of those edges (i.e. adding flow along that path). Since each augmenting path saturates at least one edge, we do at most m augmentations. Each path has length at most n. Thus the total amount of work pushing these edges to the stack, and removing them from the stack upon augmentation, and deleting saturated edges, can be bounded by $O(mn)$. Now consider the work spent pushing edges onto the stack which are later deleted because we "retreat" after not finding an s-t. An edge can only be pushed onto the stack once in this way, since it is then deleted. So the total amount spent pushing edges to the stack and deleting

them this way is $O(m)$. □

Lemma 4.2. *For unit capacity graphs,* FindBlockingFlow *returns a blocking flow in $O(m)$ time. Hence Dinic's algorithm runs in $O\left(\min\{m^{3/2}, mn^{2/3}\}\right)$ time on unit capacity graphs.*

Proof. When our depth-first search traverses some edge (u, v), one of two things will happen: either we find no augmenting path in the subtree of v, leading to the erasure of this edge, or we find an augmenting path which will necessarily saturate (u, v), again leading to its erasure. This means each edge will be traversed at most once by the depth-first search. □

Another interesting bound on the runtime, which we will not prove here, is that Dinic's algorithm will run in $O\left(E\sqrt{V}\right)$ time on bipartite matching graphs.

5 Minimum cut as a linear program

Finally we show that minimum cut may be formulated as a linear program. For a subset $S \subseteq V$ write $c_G(S)$ for the sum of $\boldsymbol{c}(e)$ over all edges $e = (u, v)$ that cross the cut, i.e. $u \in S, v \notin S$. Note that 'reverse' edges are not counted in this cut. The minimum cut problem asks us to find some $S \subseteq V$ such that $s \in S$ and $t \notin S$ such that $c_G(S)$ is minimal.

$$\begin{aligned} \min_{S \subseteq V} \quad & c_G(S) \\ s.t. \quad & s \in S \\ & t \notin S \end{aligned} \tag{2}$$

We claim this is equivalent to the following minimization problem:

$$\begin{aligned} \min_{\boldsymbol{x} \in \mathbb{R}^V} \quad & \sum_{e \in E} \boldsymbol{c}(e) \max\left\{\boldsymbol{b}_e^\top \boldsymbol{x}, 0\right\} \\ s.t. \quad & \boldsymbol{x}(s) = 0 \\ & \boldsymbol{x}(t) = 1 \\ & \mathbf{0} \leq \boldsymbol{x} \leq \mathbf{1} \end{aligned} \tag{3}$$

Recall $\boldsymbol{b}_e^\top \boldsymbol{x} = \boldsymbol{x}(v) - \boldsymbol{x}(u)$ for $e = (u, v)$. We can rewrite this as proper linear program by introducing extra variables for the maximum:

$$\begin{aligned} \min_{\boldsymbol{x} \in \mathbb{R}^V, \boldsymbol{u} \in \mathbb{R}^E} \quad & \boldsymbol{c}^\top \boldsymbol{u} \\ s.t. \quad & \boldsymbol{b}_{s,t}^\top \boldsymbol{x} = 1 \\ & \mathbf{0} \leq \boldsymbol{x} \leq \mathbf{1} \\ & \boldsymbol{u} \geq \mathbf{0} \\ & \boldsymbol{u} \geq \boldsymbol{B}^\top \boldsymbol{x} \end{aligned} \tag{4}$$

And, in fact, we'll also see that we don't need the constraint $\mathbf{0} \leq \boldsymbol{x} \leq \mathbf{1}$, leading to the following simpler program:

$$\begin{aligned} \min_{\boldsymbol{x} \in \mathbb{R}^V, \boldsymbol{u} \in \mathbb{R}^E} \quad & \boldsymbol{c}^\top \boldsymbol{u} \\ s.t. \quad & \boldsymbol{b}_{s,t}^\top \boldsymbol{x} = 1 \\ & \boldsymbol{u} \geq \mathbf{0} \\ & \boldsymbol{u} \geq \boldsymbol{B}^\top \boldsymbol{x} \end{aligned} \tag{5}$$

Lemma 5.1. *Programs* (2)*,* (4)*, and* (5) *have equal optimal values.*

Proof. We start by considering equality between optimal values of Program (2) and Program (4) Let S be an optimal solution to Program (2) and take $\boldsymbol{x} = \mathbf{1}_{V \setminus S}$. Then $\boldsymbol{x}$ is a feasible solution to Program (4) with cost equal to $c_G(S)$.

Conversely, suppose $\boldsymbol{x}$ is an optimal solution to Program (4). Let t be uniform in $[0, 1]$ and define $S = \{v \in V \mid \boldsymbol{x}(v) \leq t\}$. This is called a threshold cut. We can verify that

$$\mathbb{P}\left[e \text{ is cut by } S\right] = \max\{\boldsymbol{b}_e^\top \boldsymbol{x}, 0\}$$

and hence $\mathbb{E}_t[c_G(S)]$ is exactly the optimization function of 3 (and hence 4). Since at least one outcome must do as well as the average, there is subset $S \subseteq V$ achieving this value (or less).

We can use the same threshold cut to show that we can round a solution to Program (5) to an equal or smaller value cut feasible for Program (2). The only difference is that in this case, we get

$$\mathbb{P}\left[e \text{ is cut by } S\right] \leq \max\{\boldsymbol{b}_e^\top \boldsymbol{x}, 0\}$$

which is still sufficient. □