1 Overview

In the last lecture, we analyzed Ford-Fulkerson. In this lecture, we consider two ways to find max flows more quickly than Ford-Fulkerson.

2 Scaling Capacities

Recall that the Ford-Fulkerson algorithm has runtime $O(mf^*)$, where $f^*$ is the maximum flow. We want to improve on the factor of $f^*$. The idea is to think of each capacity $c(e) \in \{1, 2, \ldots U\}$ as a bit integer $u(e) \in \{0, 1\}^{\lceil \log_2 U \rceil}$. Then, we will “process” each $u(e)$ one bit at a time (from left to right), maintaining capacities $u'(e) = u(e)_{n(n-k)}$ (th $k$ most significant bits of $u$) and a flow $f'$ such that $f'$ is feasible on $u'(e)$. We will actually ensure that $f'$ is optimal at the beginning of each iteration. Then, when we finish, $u'(e) = u(e)$ and we will have $f' = f^*$. The complete scaling algorithm is given below. This algorithm is due to Gabow [2].

2.1 Scaling Max-Flow Algorithm

- initialize $u'(e) = 0$, $f'_e = 0$ for all $e \in E$.
- for $k = (\lceil \log_2 U \rceil - 1) \ldots 0$
  - for each $e \in E$
    * set $u'(e) = 2u'(e)$ and $f'_e = 2f'_e$ (add trailing 0 to bit vectors of $u'$, $f'$)
  - for each $e \in E$
    * set $u'(e) = u'(e) + u(e)_{k}$ (add $k$'th bit of $u$ to $u'$)
  - augment $f'$ such that it is a max flow for $G$ with capacities $u'(e)$

2.2 Analysis

Claim. At the beginning of every iteration of the scaling algorithm, $f' = (f')^*$, the maximum flow on $G$ with capacities $u'(e)$.

Proof. At the beginning of the first iteration $u'(e) = 0 \forall e \in E$ so $(f')^* = 0 = f'$. Now, suppose that $f' = (f')^*$ before we update $u'(e)$. Then, there must exist some saturated $S-T$ cut in $(G, u')$. After we double $u'(e)$ and $f'_e$, the cut is still saturated. After we add $u(e)_{k}$ to $u'$, $f'$ may not be maximal, but it is still feasible. But then we augment $f'$ so that it must equal $(f')^*$ on the new capacities.

At the $k$'th iteration of the algorithm $u'(e)$ is the $k$ most significant bits of $u(e)$, so after $\lceil \log_2 U \rceil$ iterations $u'(e) = u(e)$. Hence, by the claim, the output $f'$ of the scaling algorithm will be the maximal flow for $G$ with capacities $u'(e) = u(e)$, as desired.
The scaling algorithm runs for \(O(\log_2 U)\) iterations, each iteration dominated by the time to augment \(f'\) into a maximal flow. If we use Ford-Fulkerson, we know that this time will be \(O(mf^*)\). What can we say about \(f^*\)?

**Claim.** In each iteration of the scaling algorithm, \(f' = f^* \leq m\).

**Proof.** At the beginning to the current iteration \(f' = (f')^*\), so there must exist some saturated \(S - T\) cut of \((G, u')\). Now, as before, after we double \(u'(e), f'_e\), this cut will still have 0 capacity. After we set \(u'(e) = u(e)_k\), we can have increased \(u'(e)\) by at most 1. Since the cut cannot contain more than \(m\) edges, its residual capacity is \(\leq m\) now, so the minimum cut in \((G, u')\) must have capacity \(\leq m\) which implies that \(f^* \leq m\) (by the max-flow min-cut theorem).

Hence, if we use Ford-Fulkerson to augment \(f'\), each iteration takes \(O(m + mf^*) = O(m^2)\) steps, so we can achieve a runtime of \(O(m^2 \log U)\) for maximum flow using the scaling algorithm. Hence, the scaling algorithm is weakly polynomial.

### 3 Blocking Flow

Suppose we have a residual graph \(G_f\). For any \(v \in V\), let its level \(\ell_v\) denote the length of the shortest path distance from \(s\) to \(v\) in \(G_f\).

**Definition.** An edge \((u, v) \in E\) is admissible if \(\ell_v = \ell_u + 1\). An \(s\)-\(t\) flow on \(G_f\) is admissible if it traverses only admissible edges.

**Definition.** The level graph \(L\) associated with \(G_f\) is the subgraph of \(G_f\) comprising only its admissible edges.

**Definition.** A blocking flow \(f\) is an admissible \(s\)-\(t\) flow that saturates at least one edge in every other admissible flow \(f'\).

**Example.** Consider the following unit-capacity residual graph.

The blue flow is a blocking flow but clearly not a max flow. On the other hand, by definition, every admissible max flow is a blocking flow. Moreover, all edges here except for the dotted one are in the associated level graph.

### 3.1 A new algorithm for max flow

Recall that one of the shortcomings of Ford-Fulkerson is that the algorithm can run up to \(f^*\) iterations. We will see how blocking flows remedy this problem.

**Lemma 3.1.** Augmenting along a blocking flow strictly increases \(d(s, t)\) in the new level graph.
Proof. Let $L'$ denote the level graph after augmenting along a blocking flow. Since we augmented along a flow in $L$, it is clear that

$$L' \subseteq L \cup \text{rev}(L) \cup (G_f \setminus L),$$

where $\text{rev}(L)$ denotes the set comprising the edges in $L$ reversed. But this implies that $d_{L'}(s, t) \geq d_L(s, t)$. However, if $d_{L'}(s, t) = d_L(s, t)$, then $L$ and $L'$ must share some $s$-$t$ path, which contradicts the fact that we constructed $L'$ from $L$ by augmenting along a blocking flow. \qed

Why is this useful? Consider the following algorithm, given by Dinitz in [1].

1. Initialize $f$ to 0.
2. While there exists an $s$-$t$ path in $G_f$, augment $f$ along a blocking flow.

It is clear that when we terminate the process, we have found a max flow.

**Observation 1.** Dinitz’s runtime is given by $(\# \text{ iterations}) \times (\text{time to find blocking flow})$.

Lemma 3.1 implies that the first term is at most $n - 1$. We will consider the second term below.

### 3.2 Finding a blocking flow

We do so by running a modified version of DFS on $L$ from $s$ to build a blocking flow $f \in \mathbb{R}^m$.

We initialize $f$ to 0 and $H$ to $L$. From our current vertex $v$, we may run three kinds of moves.

- **advance($v$, $w$):** move from $v$ to $w$ for $(v, w) \in L$,
- **retreat($u$, $v$):** if $\nexists (v, w) \in L$, return its ancestor $u$ in the call stack and delete $(u, v)$ from $H$,
- **augment:** if $v = t$, augment $f$ along the minimum residual flow on the found $s$-$t$ path and delete saturated edges.

#### 3.2.1 Runtime analysis

**Claim.** In unit capacity graphs, Section 3.2’s algorithm considers each edge $O(1)$ times, which implies $O(m)$ time to find a blocking flow.

**Proof.** Observe that the algorithm

- advances at most once along any edge because $L$ is a DAG by construction,
- retreats at most once along any edge because we delete it afterwards,
- augments at most once along any edge because one augmentation immediately saturates it.

It follows that the algorithm processes each edge only a constant number of times. \qed

Then Observation 1 implies the following.

**Corollary.** In unit-capacity graphs, Dinitz’s algorithm runs in $O(mn)$ time.

Note that in unit-capacity graphs, the cut crossing the edges from $s$ encounters at most $n - 1$ units of flow. Then min-cut implies that $f^* \leq n - 1$, so Ford-Fulkerson runs in $O(mn)$ time.

It seems like our new algorithm exhibits no improvement. But it turns out that it guarantees better performance on unit-capacity graphs.
Claim. In unit-capacity graphs, Dinitz’s algorithm runs $O(\min\{\sqrt{m}, n^{2/3}\})$ iterations of finding blocking flows.

Proof. \(\sqrt{m}\)-bound. Lemma 3.1 implies that \(d(s, t) \geq k\) after \(k\) iterations of finding blocking flows. Unit capacity implies that any \(s-t\) flow can be decomposed into an edge-disjoint collection of \(s-t\) paths, so at most \(\frac{m}{k}\) \(s-t\) path remain. Therefore, the residual max flow (equivalently, the number of iterations) is at most \(\frac{m}{k}\). This means that the total number of iterations is at most \(k + \frac{m}{k}\), which equals \(2\sqrt{m}\) when \(k = \sqrt{m}\).

\(n^{2/3}\)-bound. In \(L\), define \(D_i := \{v \in V : \ell_v = i\}\). Suppose that we have already run \(k \geq 2n^{2/3}\) iterations, so \(0 \leq i \leq k\). Since at most half of the \(D_i\) contain at least \(n^{1/3}\) vertices, it follows that there must exist consecutive sets \(D_i, D_{i+1}\) such that both contain at most \(n^{1/3}\) vertices. Therefore, the number of edges from \(D_i\) to \(D_{i+1}\) is at most \(|D_i| \cdot |D_{i+1}| \leq n^{2/3}\). This means that the cut crossing these edges encounters at most \(n^{2/3}\) units of flow. Then min-cut implies that the remaining flow is at most \(n^{2/3}\), so we run at most \(2n^{2/3} + n^{2/3} = O(n^{2/3})\) iterations in total.

Corollary. In unit-capacity graphs, Dinitz’s algorithm runs in $O(m \min\{\sqrt{m}, n^{2/3}\})$ time.

3.3 General capacity graphs

We run the same algorithm, although the runtime now differs.

- We retreat at most once along each edge, so retreats contribute \(O(m)\) time.
- We saturate at least one edge each time we augment, so we augment at most \(m\) times. The length of an \(s-t\) path is at most \(n\), so augmentations contribute \(O(mn)\) time.
- We can advance at most \(n\) times before retreating, so advances contribute \(O(mn)\) time.

This implies the following.

Claim. In general-capacity graphs, Dinitz’s runs \(O(mn)\) iterations of finding blocking flows.

Corollary. In general-capacity graphs, Dinitz’s runs in \(O(mn^2)\) time.

It is also possible to show the in general capacity graphs, Dinitz’ algorithm has runtime \(O(mn + nf^*)\). Then since \(f^*\) in each round of scaling is at most \(m\) (since the previously saturated minimum cut now has at most \(m\) units of added capacity), each round of scaling when using Dinitz’ algorithm would take \(O(mn)\) time. Thus using the capacity-scaling approach with Dinitz’ algorithm used in each iteration would yield an overall runtime of \(O(mn \log U)\).

References
