COMPSCI 532: Design and Analysis of Algorithms	September 4, 2015
Lecture 4	
Lecturer: Debmalya Panigrahi	*Scribe: Allen Xiao*

1 Overview

In this lecture we will introduce the current fastest strongly polynomial algorithm for maximum flow, the push-relabel algorithm (due to Goldberg and Tarjan [GT88]). With careful data structures, it can be shown to run in $O(mn\log n)$ time. However, we will only show an analysis for $O(mn^2)$ time.

2 Preflows

Contrary to the previous algorithms based on augmentation, the push-relabel algorithm uses a new paradigm for constructing the optimal flow. The augmentation algorithms maintain a *feasible flow* and raise the flow value until s cannot reach t (no more augmenting paths). Preflow algorithms, on the other hand, begin with s unable to reach t in G_f in an *infeasible flow* and gradually modify it to make it feasible. The infeasible flow maintained is a *preflow*.

Definition 1. *$f : E \to \mathbb{R}$ is a* ***preflow*** *if for each edge (v,w) it satisfies:*

1. *Capacity constraints:*
$$f(v,w) \le u(v,w)$$
2. *Balance:*
$$\sum_{x\in V} f(x,v) \ge 0 \qquad \forall v \ne s,t$$
Contrast to flow, where this was met with equality. We allow more flow can enter v than exits, and call this difference the ***excess*** *at v, $e_f(v)$.*
3. *Skew symmetry:*
$$f(v,w) = -f(w,v)$$

If the excess at every non-source/sink vertex is 0, the preflow is a valid flow.

Definition 2. *The* ***residual network*** *G_f for a preflow is defined the same way as for flows:*

$$u_f(v,w) = u(v,w) - f(v,w)$$

Definition 3. *The* ***value*** *of a preflow is the outgoing flow from s:*

$$|f| = \sum_{v\ne s,t}\sum_{w} f(v,w) = \sum_{v} f(s,v)$$

"Flow out of s either ends up at t or gets stuck at some other vertex."

In the same way as augmentations in flows, we can add these preflows on G_f with f for a feasible preflow. The proof is the same.

3 Push-Relabel Algorithm

The push-relabel algorithm maintains a preflow and routes excess (one edge at a time) from the graph interior to s,t. Like the previous flow algorithms, it also maintains some notion of *admissible* edges, and moves flow only on these. This is maintained through an explicit *height function* over vertices.

Definition 4. *Push-relabel maintains a **height function** $h(v) \geq 0$ over all vertices. We call an edge (v,w) **admissible** if:*

$$h(v) > h(w)$$

"Sending flow on admissible edges is like sending flow downhill."

In contrast to the distances from t ($d(v)$) we used in blocking flow to define admissibility, $h(v)$ is defined and updated explicitly by the algorithm (and not a naturally ocurring quantity in the graph). $h(v)$'s relationship with the actual G_f distances $d(v)$ will become more clear as we prove more limits on $h(v)$.

The algorithm starts by initializing the preflow such that its value is a simple upper bound for any feasible flow: it saturates the outgoing edges (degree cut) of s. Then, the algorithm manipulates the preflow and heights using only two operations, to eventually move all excesses to s or t.

Definition 5. *The two operations:*

1. PUSH(v,w)*: For some r we choose, this function is applicable if $e_f(v) \geq r$ and $u_f(v,w) \geq r$. The push moves r excess across (v,w), in other words:*

$$\begin{aligned} e_f(v) &\leftarrow e_f(v) - r \\ u_f(v,w) &\leftarrow u_f(v,w) - r \\ e_f(w) &\leftarrow e_f(w) + r \end{aligned}$$

   In our case we will always try to send the greatest r we can:

$$r = \min\{e_f(v), u_f(v,w)\}$$

   *Whenever $r = u_f(v,w)$, we call this a **saturating** push. Otherwise, the push is a **non-saturating** push.*

2. RELABEL(v)*: We can relabel v when it has positive excess, but no outgoing admissible edges. This raises $h(v)$ (the label) until v has at least one admissible outgoing edge.*

$$h(v) \leftarrow \min_{w:(v,w)\in E} \{h(w)\} + 1$$

The algorithm itself is very simple:

Algorithm 1 (Push-Relabel 1988)

$f \leftarrow 0$
$h(v) \leftarrow 0 \quad \forall v \in V \setminus \{s\}$
$h(s) \leftarrow n$
Saturate all edges leaving s.
while $e_f(v) > 0$ for any $v \neq s,t$ **do**
    if v has and admissible outgoing edge (v,w) **then**
        PUSH(v,w)
    else
        RELABEL(v)
    end if
end while

3.1 Correctness

When no internal vertices have excess, the preflow is a feasible flow. If we prove that the preflow maintains t unreachable from s in G_f, then it must also be a maximum flow. We will prove that the algorithm maintains a preflow where t is always unreachable from s in G_f.

Lemma 1. *The labels (heights) satisfy the following properties throughout the algorithm:*

1. $h(s) = n, h(t) = 0$
2. *For any* $(v,w) \in G_f$, $h(v) \leq h(w) + 1$.

Proof. We can prove this by induction on the operations. Initially, both conditions are met – $h(s), h(t)$ are initialized as such and all residual edges have $h(v) - h(w)$ equal to 0 or $-n$. Suppose that the conditons are met at any point, then apply either operation:

1. PUSH(v,w): This may add a reverse edge $(w,v) \in G_f$. Push only works on edges where $h(v) > h(w)$, so by our assumption it must be that $h(v) = h(w) + 1$. The reverse edge (w,v) therefore has

$$h(w) = h(v) - 1 \leq h(v) + 1$$

   And the assumptions are satisfied for the new edge.

2. RELABEL(v): By the conditions for relabel require that v has no admissible edges, i.e. for all edges (v,w):

$$h(v) \leq h(w)$$

   Then, relabeling v gives:

$$h(v) \leq h(w) + 1$$

   for all neighboring w. Relabel is never applied to s,t, so the first condition is also maintained.

□

The second point of Lemma 1 is what ties our purposefully specified heights $h(v)$ to graph distances $d(v)$. Effectively:

$$h(v) \leq d(v)$$

Heights are a *lower bound* to distances from t. As we show in the next lemma, $h(v) = n$ directly implies that $d(s) = \infty$, and so s cannot reach t in the residual graph.

Lemma 2. *There is no s-t path in G_f during the push-relabel algorithm.*

Proof. Suppose for contradiction such a path exists. Then there is a path $s, v_1, \ldots, v_k, t$ in G_f. By the first condition of Lemma 1,

$$\begin{aligned} h(s) &= n \\ h(t) &= 0 \end{aligned}$$

By the second condition, we must have:

$$h(s) \leq h(v_1) + 1 \leq \cdots \leq h(t) + (n-1)$$

And we find a contradiction:

$$n \leq n - 1$$

Therefore, there must be no s-t path in G_f. □

The statement of this lemma seems to imply that the algorithm maintains some sort of s-t cut which shrinks into the minimum cut. The algorithm even begins by creating an s-t cut: the degree cut of s. After proving Lemma 2, we conclude that f is a maximum flow.

3.2 Running Time

We will bound the number of times RELABEL is invoked, the number of saturating PUSH, and the number of non-saturating PUSH.

First, we we prove an upper bound on the height of any vertex.

Lemma 3. *If v has positive excess, then there is a v-s path in G_f.*

Proof. Let S be the set of vertices with a path to s in G_f.

$$S = \left\{ v \mid \text{there is a } v\text{-}s \text{ path in } G_f \right\}$$

By definition, no edges in $(S, V \setminus S)$ have positive residual capacity. All residual edges go from S to $V \setminus S$, meaning the flow moves the opposite direction.

$$f(S, V \setminus S) \leq 0$$

Now, add a dummy vertex t', and add residual edges (v, t) from every excess vertex, routing all excess to t'. This turns the preflow into a feasible s-t' flow. Let $T = (V \setminus S) \cup \{t'\}$. Since (S, T) is a s-t' cut:

$$f(S, T) = \sum_{v \in V} f(s, v) = \sum_{v \in V} e_f(v)$$

Combining, we have:

$$f(S, \{t'\}) \geq \sum_{v \in V} e_f(v)$$

Since the only edges to t' are the ones carrying excess from the original preflow, it must be that the bound above is met with equality and *all* excess vertices lie in S. The lemma follows by definition of S. □

Lemma 4. *For every vertex v, $h(v) \leq 2n-1$.*

Proof. For v to be relabeled, it must have $e_f(v) > 0$. By Lemma 3, there must be a G_f path $v, v_1, \ldots, v_k = s$. Applying Lemma 1 over this path:

$$h(v) \leq h(v_1)+1 \leq \cdots \leq h(s)+k \leq h(s)+(n-1) \leq 2n-1$$

□

Corollary 5. *The number of relabel operations is $\leq 2n^2$.*

Next, we look at pushes.

Lemma 6. *There are at most n pushes over any edge which saturate it.*

Proof. A push is saturating if, after the push, $u_f(v,w) = 0$. Any push has $h(v) > h(w)$, so between two saturating pushes on (v,w) $h(v)$ must increase by at least 2. By Lemma 4, $h(v) \leq 2n-1$. The number of saturating pushes over (v,w) is therefore no more than n. □

Corollary 7. *The number of saturating pushes is $\leq 2mn$.*

Lemma 8. *The number of non-saturating pushes is $\leq 5mn^2$*

Proof. Consider the following potential function:

$$\phi = \sum_{v \neq s,t : e_f(v) > 0} h(v)$$

We know that:

$$\begin{aligned} \phi_{\text{init}} &= 0 \\ \phi_{\text{final}} &= 0 \end{aligned}$$

The final potential is also 0, as all excess ends in t.

Each relabel raises potential by at least 1 (increments one $h(v)$). Applying Corollary 5:

$$\text{for relabels:} \qquad \Delta\phi \leq 2n^2$$

A saturating push on (v,w) can raise potential by at most $2n-2$, if w previously had no excess. Applying Corollary 7:

$$\text{for saturating pushes:} \qquad \Delta\phi \leq 4mn^2$$

Finally, a non-saturating push on (v,w) *lowers* potential As $h(v) > h(w)$, Lemma 1 tells us how much:

$$\text{for a single non-saturating push:} \qquad \Delta\phi \leq h(w) - h(v) = -1$$

Combining, we can get the total number of non-saturating pushes. Let T be the number of non-saturating pushes.

$$\begin{aligned} 0 &\leq 0 + 2n^2 + 4mn^2 + T(-1) \\ T &\leq 2n^2 + 4mn^2 \leq 5mn^2 \end{aligned}$$

□

Now that we have the operation counts, runtime is simple.

Theorem 9. *The running time of the push-relabel algorithm is $O(mn^2)$.*

Proof. Each of the operations (push, relabel) can be done in constant time. Adding the operation counts gives us $O(mn^2)$ time. □

4 Summary

We can get a strong improvement in push-relabel by improving our choice of vertices and edges for pushes and relabels. If we always push flow out of the vertex with maximum $h(v)$, we can prove a running time of $O(n^2\sqrt{m})$. A version using certain dynamic trees and performing simultaneous pushes runs in $O(mn\log\frac{n^2}{m})$ time. This is also used by the best algorithm to find a blocking flow, which has a similar time bound. This is the fastest strongly polynomial maximum flow algorithm, and the most commonly used max-flow algorithm in practice.

References

[GT88] Andrew V Goldberg and Robert E Tarjan. A new approach to the maximum-flow problem. *Journal of the ACM (JACM)*, 35(4):921–940, 1988.