CS491G Combinatorial Optimization
Push-Relabel Maximum Flow Algorithms
Lecture Notes

Zi Zhuang

1. Motivation and Overview

The Goldberg and Tarjan algorithm introduced the push-relabel method for finding the maximum flow. In the previous lecture we introduced the max-flow algorithm. There are cases through, where this algorithm does not have good performance. Consider a graph with a many-hop, high-capacity path from the source, r, to another node, v, where connected to the sink, s, by many low-capacity paths. In this topology, the max-flow algorithm introduced earlier must send single units of flow individually over the sequential part of the path (from r to v), since no full path from r to s has a capacity of more than one. This is clearly a fairly time-consuming operation, and we can see that the algorithm would benefit from the ability to push a large amount of flow from r to v in a single step. This idea gives the intuition behind the Push-Relabel algorithm. Push-relabel involves examining vertices with positive flow excesses and pushing flow from them to vertices that are estimated to be closer to the sink.

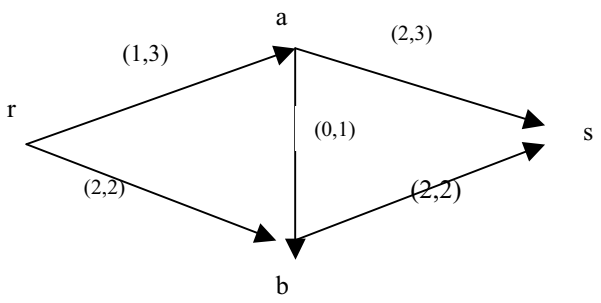
2. Description of Framework

For purposes of this section it is convenient to take $u_{wv} = x_{wv} = 0$ if $vw \in E$ and $wv \notin E$.

For a vector $\vec{x}$ satisfying $0 \le x \le u$, we define the auxiliary digraph $G(\vec{x})$ just as though $\vec{x}$ were a feasible flow, except that we do not bother with parallel arcs. That is, we put $vw \in E(G(x))$ if and only if $x_{wv} > 0$ or $x_{vw} < u_{vw}$.

If vw is an arc of $G(\vec{x})$, then up to $\tilde{u}_{vw} = u_{vw} - x_{vw} + x_{wv}$ units of flow can be "pushed" from v to w, without violating any non-negativity or capacity restrictions. (We call $\tilde{u}_{vw}$ the residual capacity of vw.) Namely, we can increase x_{vw} to u_{vw} and we can decrease x_{wv} to 0.

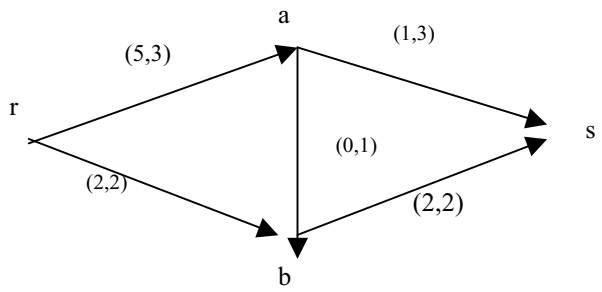
Definition: We call $\vec{x}$ a *preflow* if it satisfies $f_{\vec{x}}(v) \ge 0$ for all $v \in V \setminus \{r, s\}$.



? Is a feasible preflow a flow?

NO, Flow should be equate to 0. Here for vector a, income flow 1 and outgoing flow 2 is 1-2= - 1

Definition: It is a *feasible* preflow if it also satisfies $0 \le x \le u$.



If we choose the pair (v,w) so that $\widetilde{u}_{vw} > 0$ and $f_x(v) > 0$, then pushing up to $\varepsilon = \min(\widetilde{u}_{vw}, f_x(v))$ from v to w will produce a new feasible preflow. This is what we mean by doing a push on vw. (There is some ambiguity if both vw and wv are arcs of G and $\varepsilon < \widetilde{u}_{vw}$. We can resolve it by decreasing x_{wv} as much as possible. That is, we decrease x_{wv} by $\varepsilon' = \min(\varepsilon, x_{wv})$, and we increase x_{vw} by $\varepsilon - \varepsilon'$.)

Definition: We call a node $v \in V \setminus \{r,s\}$ *active* if it has positive net flow, that is, if $f_{\vec{x}}(v) > 0$.

So a preflow is a flow precisely if there are no active nodes.

Definition: We say that a vector $\vec{d} \in (Z_+ \cup \{\infty\})^{|v|}$ is a *valid labelling* with respect to a preflow $\vec{x}$ if

$$d(r) = n, d(s) = 0 \qquad (1.1)$$

for every arc vw of $G(\vec{x})$, $d(v) \le d(w) + 1$.

Notice that $d(v) = d_x(v,s)$ satisfies all of these conditions except $d(r) = n$. (Recall that $d_{|x|}(v,w)$ denotes the number of arcs in a shortest (v,w)-dipath in $G(\vec{x})$). So we can "almost" get a valid labelling for any feasible preflow.

Corollary: Not every feasible preflow admits a valid labelling

But it is easy to construct some feasible preflow and a valid labelling for it, using the following procedure, which we call initialize $\vec{x}, \vec{d}$. Put $x_e = u_e$ for all arcs e having tail r, and put $x_e = 0$ for all other arcs $e \in E$. Put $d(r) = n$ and $d(v) = 0$ for all other nodes $v \in V$. It is easy to check that x and d do satisfy (1.1). (Remark: We must assume that each $u_{rv} \neq \infty$.) The existence of a valid labelling for a preflow implies an important property of the preflow, that is "saturates a cut."

Lemma.1: If $\vec{x}$ is a feasible preflow and $\vec{d}$ is a valid labelling for $\vec{x}$, then there exists an (r,s)-cut $\delta(R)$ such that $x_{vw} = u_{vw}$ for all $vw \in \delta(R)$ and $x_{vw} = 0$ for all $vw \in \delta(\overline{R})$.

Proof: Since there are n nodes, there exists a value k, such that $0 < k < n$ for all $v \in V$. Take $R = \{v \in V : d(v) > k\}$. Then $r \in R$ and $s \notin R$. Clearly, (1.1) implies that no arc of $G(\vec{x})$ leaves R, which implies the statement of the proposition.

Corollary: If a feasible flow $\vec{x}$ has a valid labelling, then $\vec{x}$ is a maximum flow.

The corollary gives a possible termination condition for a maximum flow algorithm. A push-relabel algorithm maintains a feasible preflow and a valid labelling (and thus by Lemma.1, a saturated cut) and terminates when the preflow becomes a flow. So there is certain duality with an augmenting path algorithm, which maintains a feasible flow and terminates when a cut becomes saturated.

Next we show in what sense a valid labelling gives an approximation to distances in $G(\vec{x})$.

Lemma.2 For any feasible preflow $\vec{x}$, and any valid labelling $\vec{d}$ for $\vec{x}$, we have

$$d_{\vec{x}}(v,w) \geq d(v) - d(w), \text{ for all } v, w \in V$$

Proof: If $d_x(v,w) = \infty$ this is certainly true, so suppose $d_{\vec{x}}(v,w)$ is finite, and consider any shortest (v,w) dipath in $G(\vec{x})$. Adding up the inequality $d(p) - d(q) \leq 1$ on the arcs pq of the dipath gives the result.

In particular, it follows from Lemma.2 that $d(v)$ is a lower bound on $d_{\vec{x}}(v,s)$, and $d(v) - n$ is a lower bound on $d_{\vec{x}}(v,r)$. Notice that if $d(v) \geq n$, this means that $d_{\vec{x}}(v,s) = \infty$, and excess flow at v should be moved toward the source r. Whether $d(v)$ is large or small, we try to move flow toward nodes w having $d(w) < d(v)$, since such nodes are estimated to be closer to the ultimate destination. Moreover, by the definition of valid labelling, and vw an arc of $G(\vec{x})$ implies that $d(w) = d(v) - 1$. Therefore, push is applied only to arcs vw of $G(\vec{x})$ such that v is active and $d(v) = d(w) + 1$. Such arcs are

called *admissible*. Notice that d is still a valid labelling for the new preflow, since the only (possible) new restriction arises if wv becomes an arc of $G(\vec{x})$; this would require $d(w) \le d(v) - 1$, which is already satisfied.

Now suppose that v is active but there is no arc vw of $G(\vec{x})$ with $d(v) = d(w) + 1$. Then we can increase $d(v)$ to $\min(d(w) + 1 : vw \in E(G(\vec{x})))$, without violating the validity of the labelling. This is the relable operation.

Namely, once an active node v is chosen, we continue to perform push operations on admissible arcs vw of $G(\vec{x})$ until v either becomes inactive or is relabeled. Notice that this is possible, because if there are no admissible arcs vw and v is still active, then v can be relabeled. To perform this sequence of operations is to process v.

Process v
While there exists an admissible arc vw
Push on vw;
If v is active
Relabel v.

Now we can state the *push-relabel* maximum flow algorithm quite simply.

Push-Relabel Algorithm

Initialize $\vec{x}, \vec{d}$;
While $\vec{x}$ is not a flow
Choose an active node v;
Process v.

The basic step of the push-relabel algorithm for the maximum flow problem is to choose an active node v, then choose an arc vw of $G(\vec{x})$, and do a push on vw. However, we need to specify more carefully the choice of vw. Otherwise, for example, one could easily have an infinite loop consisting of a push on vw, then a push in wv, then a push on $vw, \ldots$. The additional restriction comes from the idea that we want, as much as possible, to push flow toward the sink, s. However, it is quite possible that we reach a point where no more flow can be pushed toward the sink, but there are still active nodes. In this case the only way to restore conservation of flow is to push the excess back toward the resource, r. The device that allows us to make decisions about the direction of pushes is an estimate of distance in $G(\vec{x})$.

We can summarize its execution as follows:

1. Set the source label $d(r)$ =n, the sink label to $d(s)$ =0, and the labels on the remaining nodes to 0.
2. Send out as much flow as possible from the source r, saturating its outgoing edge and placing excesses on its neighboring nodes.
3. Calculate the residual edges.
4. Relabel the active nodes, increasing values as much as possible without violating the label constraint.
5. Push as much flow as possible on some admissible edge.
6. Repeat steps 4 and 5 until there are no active nodes left in the graph.

Lemma.3: If $\vec{x}$ is a preflow and w is an active node, then there is a (w,r) – dipath in $G(\vec{x})$.

Proof: Let R denote the set of nodes v for which there is a (v,r) –dipath in $G(\vec{x})$. Then no arc leaves $\overline{R}$ in $G(\vec{x})$, so $\vec{x}$ (δ(R)) = 0. But suppose that we add the inequalities f_x(v) $\geq$ 0 for $v \in \overline{R}$. Then we get x(δ(R)) - x($\delta(\overline{R})$) $\geq$ 0. With x(δ(R)) = 0, this implies x($\delta(\overline{R})$)=0. So the sum of the inequalities holds with equality, which means that each of them does. That is, there is no active node in $\overline{R}$, so $w \in R$ as required.

3. Analysis of the Algorithm

Lemma.4: At every stage of the push-relabel algorithm, for every $v \in V$, we have $d(v) \leq$ 2n-1. Each node is relabelled at most 2n-1 times, and there are O(n^2) relables in all.

Proof: Since each relabel of v increases $d(v)$ by at least 1, the second statement follows from the first. Since only active nodes relabelled, it is enough to prove the first statement for v active. By Lemma.3 $d_{\vec{x}}(v,r) \leq n-1$. By lemma.2 $d_{\vec{x}}(v,r) \geq d(v)-n$. Combining these two inequalities gives $d(v) \leq 2n-1$.

It is useful to divide the push operations into two kinds. A push on vw is saturating if $\tilde{u}_{vw} \leq f_{\vec{x}}(v)$, so that the value pushed is $\tilde{u}_{vw}$, and arc vw leaves $G(x)$. Otherwise, the push is nonsaturating, and in this case is no longer active.

Lemma.5 The number of saturating pushes performed by the push-relabel algorithm is at most 2mn.

Proof: Consider a fixed pair (v,w) of nodes, such that $vw \in E$ or $wv \in E$. Between two saturating pushes on vw, there must be a push on wv, since otherwise vw is not an arc of $G(\vec{x})$. But since $d(v) = d(w)+1$ for a push on vw, and $d(w) = d(v)+1$ for a push on wv, and since $d(v)$ never decreases, there must be a relabel of w before there can be a push on

wv. Hence between any two saturating pushes on vw, $d(w)$ increases by at least 2, and this can happen, by Lemma.4, at most n. Therefore, the total number of saturating pushes associated with an arc $vw \in E$ (that is on vw, or wv) is at most 2n, and the total for all arcs is at most 2mn.

Lemma.6 The number of nonsaturating pushes performed by the push-relabel algorithm is $O(mn^2)$.

Proof: Let A be the set of active nodes with respect to the preflow x, and let $D=\Sigma(d(v) : v \in A)$. Observe that D is initially 0 and is never negative.

- Each relabel increases D by 2n-1.
- Saturating push on vw may increase $d(A)$ by as much as 2n-1(since w could enter A and v could remain in A).

Total increase in D = $(n-2)(2n-1) + 2mn(2n-1) = O(mn^2)$

If $w \in A$, you could lose D to 2n-1.
If $w \notin A$, you will lose 1.
Every nonsaturating push decreases D by at least 1, Since the total decrease in D is at most the total increase, there are $O(mn^2)$ nonsaturating pushes.

Lemma.7 The maximum distance push-relabel algorithm performs $O(n^3)$ nonsaturating pushes.

Proof: Any nonsaturating push from a node v makes v inactive, and v cannot become active again before there is a relabel, since all active nodes w have $d(w) \leq d(v)$. Hence, if there are n nonsaturating pushes is less than n times the number of labels, so by Lemma.4, it is $O(n^3)$.

Theorem 1: The push-relabel algorithm performs $O(n^2)$ relabels and $O(mn^2)$ pushes.

Theorem 2: The maximum distance push-relabel algorithm performs $O(n^2)$ relabels and $O(n^3)$ pushes.

4. Implementation of Push-Relabel Algorithms

We have proved quite good bounds on the numbers of basic steps of the push-relabel algorithms. In order to convert these into statements about running times, we need to give some details of implementation. For each node v we keep a list L_v of the pairs vw or wv (or both) is an arc of G. We may refer to these as arcs; actually, they are the possible arcs of $G(\vec{x})$. With each element vw of L_v we keep $\tilde{u}_{vw}$. We also keep links between the pairs

$vw \in L_v$ and $wv \in L_v$, so that after a push on vw, we can update fixed. In addition, we keep with each node v the values $d(v)$ and $f_{\bar{x}}(v)$.