ICS 260—Fall, 2000–Class Notes 5: The Preflow-Push Algorithm

Notes Originally Developed by Sandy Irani
Modified by Michael B. Dillencourt

This material is taken from [AMO] 7.6; also see [CLR] 27.4.

1 Intuition

An inherent bottleneck in the augmenting paths is the time to push flow along each augmenting path. This will be $O(n)$ time in the worst case for each iteration. Preflow-push algorithms do not suffer from this problem and get dramatic improvements in both the worst case bounds and the practical performance of the algorithm.

- Example

The augmenting path algorithms always maintain a feasible flow and try to improve the flow to achieve an optimal one. The preflow-push algorithms obtain optimality right away, but must adjust the flows to attain a feasible flow. Preflow-push algorithm generally perform better than the augmenting path algorithms, both in terms of the asymptotic worst-case complexity and in practice.

2 Some Definitions

The current 'flow' maintained by the preflow-push algorithms do not always maintain the balance constrains at each node. This flow is referred to as a *preflow*. The condition that is maintained is that the flow into a vertex is at least the amount out of a vertex:

$$e(v) = \sum_{(u,v)} f(u,v) \geq 0, \qquad \text{for all} \quad v \neq s, t.$$

$e(v)$ is said to be the *excess* at node v. Since no edges leave t, $e(t) \geq 0$. Thus, node s is the only node with a negative excess. A node which is neither the source nor the sink which has a strictly positive excess is said to be an *active* node.

The algorithm will also maintain a set of distance labels on the nodes. These definitions are also used in the shortest path augmentation algorithm. $d[v]$ will denote the distance label for node v. The labels will always obey the following two conditions:

1. $d[t] = 0$.
2. $d[i] \leq d[j] + 1$ for each arc (i,j) in G_f.

Fact 1 *If the two conditions above are maintained, then $d[v]$ is always a lower bound on the length of the shortest path from v to t in G_f.*

Fact 2 *If $d[s] \geq n$, then there is no path from s to t in G_f.*

An arc (u, v) is said to be *admissible* if $d(u) = d(v) + 1$. A path is said to be admissible if it is composed entirely of admissible edges.

Fact 3 *An admissible augmenting path from x to t is a shortest path from x to t.*

Proof. The fact that it is an augmenting path follows from the fact that each edge (u, v) is in G_f and must have $c_f(u, v) > 0$. Since $d[u] = d[v] + 1$ for each edge in the path, $d[x]$ must be the length of the path. $d[x]$ is also a lower bound on the length of the shortest path from x to t. Thus, the path must be the shortest path from x to t. ∎

3 The Algorithm

The goal is to push flow towards the sink. The distance labels are used to determine in which directions to push flow: flow is only pushed from active vertices along admissible arcs. If there is an active vertex with no admissible arc leaving it, its distance label is raised so that it will have an admissible arc. Figure 3 gives the generic Preflow-Push Algorithm.

Note that the initialization ensures that the distance labels are valid. The $d[v]$ for $v \neq s$ values are the true distance values from v to t. Also, since each arc out of s is saturated, there are no edges (s, v) in G_f, so we do not have to worry about whether $d[s] \leq d[v] + 1$ for any node v. It also ensures that all the nodes adjacent to s are active, so we can start the algorithm by finding an active node. Furthermore, since $d[s] \geq n$, there is no path from s to t in the residual graph. This means that the preflow is at least the value of the maximum flow. Since the distance labels never decrease, this will always be the case, so when the algorithm terminates with a valid flow, it will be the maximum flow.

4 Running Time of the Preflow-Push Algorithm

The first thing we must establish is that the distance values are always valid. First of all $d[t]$ never increases since it is never active and we never call Push-Relabel in t. Secondly, flow is only pushed along admissible arcs and a node is only relabeled when no admissible arc leaves that node.

The next thing that we must establish is that the algorithm never gets 'stuck' in a Push-Relabel step. That is, if a relabel is necessary, then we will not minimize over an empty set. This fact follows from the following lemma:

Lemma 4 *Any active node i has a path to node s in the residual network.*

```
Procedure PREPROCESS:
(1) f is the zero flow.
(2) Compute the exact distance labels d[v]
(3) f(s,u) = c(s,u) for each (s,u) ∈ E
(4) d[s] = n.
```

```
Procedure PUSHRELABEL(u):
(1) if the network has an admissible arc (u,v)
(2)     push δ = min{e(u), c_f(u,v)} units of flow from node u to node v
(3) else d[u] ← min{d[v] + 1 | (u,v) ∈ E and c_f(u,v) > 0}
```

```
Algorithm PREFLOWPUSH(u):
(1) Preprocess.
(2) while network contains an active node:
(3)     Select an active node u.
(4)     Push-Relabel(u)
```

Figure 1: The generic Preflow-Push algorithm.

Thus, we have established that each call to Push-Relabel results in either a Push or Relabel. Now we must bound the number of times we can push flow along an arc or relabel a vertex. We will distinguish between push operations which saturate the arc and those which do not saturate the arc.

To bound the number of relabelings, we have the following lemma:

Lemma 5 *For each node u, $d(u) < 2n$.*

Proof. The last time node u is relabeled, it has positive excess. This means that there is a path p with at most $n-2$ edges from u to s in the residual network. Each edge (x,v) in this path has $d[x] \leq d[v]+1$ and $d[s]=n$. This implies that $d[u] < 2n$. ∎

This implies that the number of relabel operations is at most $2n^2$.

Next we bound the number of saturating pushes.

Lemma 6 *The algorithm performs at most nm saturating pushes.*

Proof. This is similar to the argument we used for the shortest augmenting path algorithm. Between successive saturating pushes to an edge (u,v), both $d[u]$ and $d[v]$ must increase by at least

2. (Recall that when flow is pushed from u to v, $d[u] = d[v] + 1$. When flow is pushed back from node v to node u, $d[v] = d[u] + 1$. Then when edge (u, v) is saturated again, $d[u] = d[v] + 1$. Since the distance labels do not increase, this means that $d[u]$ and $d[v]$ have increased by at least two each).

Since $d[u] < 2n$, there can be at most n saturating pushes along edge (u, v). This means that there are a total of nm saturating pushes. ∎

Finally, we bound the number of non-saturating pushes:

Lemma 7 *The algorithm performs at most $O(n^2 m)$ non-saturating pushes.*

Proof. Let I be the set of active nodes. This argument uses a potential function which is $\Phi = \sum_{u \in I} d[u]$. The initial value of Φ is at most $2n^2$ and Φ always remains non-negative. This means that the total amount that Φ decreases is at most the amount that Φ increases plus $2n^2$.

We will prove that all the relabel steps increase Φ by a total of at most $2n^2$. Furthermore, each saturating push causes Φ to increase by at most $2n$. Since there are at most nm saturating pushes, Φ will increase by at most $2mn^2$ due to all the saturating pushes. Finally, we will show that each non-saturating push causes Φ to decrease by at least 1. Putting these facts together, we will get that the total amount by which Φ increases is at most $2n^2 + 2mn^2$ and the total amount by which Φ decreases is at most $4n^2 + 2mn^2$. Since each non-saturating push causes Φ to decrease by at least 1, there are $O(mn^2)$ non-saturating pushes.

Since each relabel step causes the distance label of a node to increase by at least 1, and the $d[u]$ for each node is at most $2n$, there can be at most $2n^2$ relabel steps.

A saturating push along edge (u, v) could create an excess at node v, which would result in an increase to Φ of $d[v]$. Since $d[v] < 2n$, this will results in an increase of at most $2n$ to Φ and a total increase of at most $2mn^2$ over all saturating pushes.

Finally consider a non-saturating push from node u to node v. The excess at node u goes to 0 and will create some excess at node v. Thus, u is removed from I and v may be added to I. Since flow is pushed along edge (u, v), it is an active edge and $d[u] = d[v] + 1$. Thus, Φ decreases by $d[u] = d[v] + 1$ and increase by $d[v]$, resulting in a net decrease of 1. ∎

5 Different Implementations

The generic Preflow-Push algorithm does not specify how to select an active node for a push-relabel operations. Different implementations have specific ways of selecting the next active vertex which result in better asymptotic running times.

We will adopt the rule that each algorithm selects a node u for *examination* in which the Push-Relabel function will be called on u until either node u is relabeled or there is no more excess at node u to push. Once a node is examined, the question is then how to select the next node for examination. Here are some possibilities:

- **First-in-first-out** FIFO order. The algorithm maintains a queue. Each new active vertex is selected from the head of the queue. If pushing flow along edge (u, v) makes v active, it is put at the tail of the queue of active nodes. If a node u is still active after it has been examined, it is put back at the tail of the queue. This implementation runs in time $O(n^3)$.
- **Highest distance label** The active vertex with the highest distance label is chosen for examination. This algorithm runs in time $O(n^2 m^{\frac{1}{2}})$.
- **Excess scaling algorithm** Flow is pushed from a node with 'sufficiently high excess' to a node with 'sufficiently low excess'. The running time of this algorithm is $O(nm + n^2 \log c_{max})$.