1 Preflow-Push Algorithms

In this lecture, we will discuss *preflow-push* algorithms for the maximum flow problem. Similar to path-augmenting algorithms, preflow-push is a general concept with different implementations. The big difference of preflow-push algorithms to path-augmenting algorithms is that they do not maintain a feasible flow in a network throughout their execution, but only a *preflow*, which is a flow that may violate flow conservation on some edges. The algorithm will push excess flow from vertices to their neighbors, and we will use a *height function* to guide this procedure. We start with a definition of a preflow and a height function.

Definition 1 (Preflow)**.** *Let $\mathcal{N} = (V, E, c, s, t)$ be an s-t-network. A* preflow *in $\mathcal{N}$ is a function $f : V \times V \to \mathbb{R}$ such that*

(i) for all $e \in E$, $0 \le f(e) \le c(e)$, and

(ii) for all $v \in V \setminus \{s\}$, $\sum_{e\in\delta^-(v)} f(e) \ge \sum_{e\in\delta^+(v)} f(e)$.

Figure 1 depicts an example of a preflow.

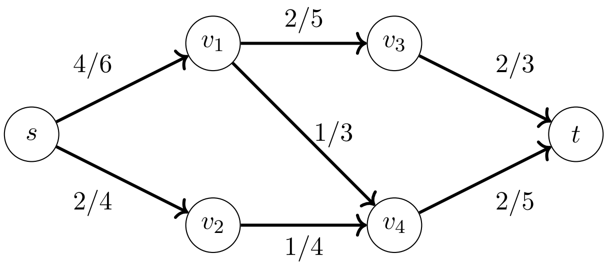

Figure 1: An example network with a preflow.

Since flow conservation may be violated at some vertex v, we can think of the *excess* at v as the amount of flow that enters v but does not leave it. Formally, we define the excess of a vertex v in a preflow f as $e_f(v) := \sum_{e\in\delta^-(v)} f(e) - \sum_{e\in\delta^+(v)} f(e)$. Note that $e_f(v) \ge 0$ for $v \in V \setminus \{s\}$, and $e_f(s) = 0$.

Definition 2 (Height function)**.** *Let $\mathcal{N} = (V, E, c, s, t)$ be an s-t-network. A* height function *is a function $d : V \to \mathbb{N}_0$. A height function d is* legal *with respect to a preflow f in $\mathcal{N}$ if*

(i) $d(s) = |V|$,

(ii) $d(t) = 0$, and

(iii) $d(u) \le d(v) + 1$ for every edge $(u, v) \in E_f$.

The idea of the height function d is similar to the idea of the distance function δ_f, which we have seen in the previous lecture. The height function is used to guide the flow of excess from vertices to their neighbors. One can think of $d(u)$ as an estimation of $\delta_f(u)$, that is, the distance of u to t in $\mathcal{N}_f$. Thus, "falling" edges (u, v) with $d(u) = d(v) + 1$ intuitively lead into the direction of t, and our algorithm will only push excess flow over such *eligible* edges.

Definition 3 (Eligible edges and active vertices)**.** *An edge $(u, v) \in E_f$ is* eligible *with respect to a preflow f and a height function d if*

(i) $e_f(u) > 0$, and

(ii) $d(u) = d(v) + 1$.

Moreover, we call a vertex $v \in V \setminus \{s,t\}$ active *if* $e_f(v) > 0$.

In the following, we will introduce operations that preflow-push algorithms can perform.

Function `Push`(u,v)**:**
    Precondition : $(u,v) \in E_f$, u is active, and (u,v) is eligible
    if $(u,v) \in E$ **then**
        $f(u,v) \leftarrow f(u,v) + \min\{e_f(u), c_f(u,v)\}$
    if $(u,v) \in \overleftarrow{E}$ **then**
        $f(v,u) \leftarrow f(v,u) - \min\{e_f(u), c_f(u,v)\}$

Function `Relabel`(u)**:**
    Precondition : u is active and no edge out of u is eligible
    $d(u) \leftarrow d(u) + 1$

An algorithm can use `Relabel` to increase the height of an active vertex if it lies in a "valley": all edges lead to higher vertices. Then, the algorithm can increase its height to push its excess to a neighbor. Initially, we saturate all edges out of s, and set the height of s to n and the height of all other vertices to 0.

Function `Init`$(\mathcal{N})$**:**
    $d(s) = n, d(v) = 0$ for all $v \in V \setminus \{s\}$
    $f \coloneqq 0$
    for $v \in V$ *with* $(s,v) \in E$ **do**
        $f(s,v) \leftarrow c(s,v)$

The generic Preflow-Push algorithm can now be stated very compact: it simply performs a `Push` or `Relabel` operation as long as possible.

Algorithm 1: The Generic Preflow-Push Algorithm

Input: An s-t-network $\mathcal{N} = (V,E,c,s,t)$
`Init`$(\mathcal{N})$
while *some* `Push` *or* `Relabel` *operation is possible* **do**
    Carry out such an operation
return f

We start our analysis with the following three lemmas, which prove the correctness of the algorithm.

Lemma 1. *After* `Init` *and after every* `Push` *or* `Relabel`*, f is a preflow and d is a legal height function with respect to f.*

Proof. It is easy to check that f is initially a preflow, and also remains a preflow after every `Push` or `Relabel` operation.

After `Init`, we have $d(s) = n$ and $d(v) = 0$ for all $v \in V \setminus \{s\}$. Thus, $d(u) = 0 \le 1 = d(v) + 1$ for all $u, v \in V \setminus \{s\}$. Moreover, all edges $(s,v) \in \delta^+(s)$ are saturated in f after `Init`, which means that $(s,v) \notin E_f$ and $(v,s) \in E_f$. Since $d(v) = 0 \le n + 1 = d(s) + 1$, the height function is legal after `Init`.

If `Relabel`(u) is executed, by the precondition (and by induction that d is legal before this operation) it must hold for every edge $(u,v) \in E_f$ that $d(u) < d(v) + 1$, and thus, $d(u) \le d(v)$.

Thus, after executing the operation, it $d(u) \leq d(v) + 1$ for every edge $(u, v) \in E_f$, and thus, d is legal after the operation.

If `Push(`u, v`)` is executed, the edge (u, v) may be removed from the residual network if it becomes saturated. Also, the reverse edge (v, u) may be added to the residual network. However, since $d(u) = d(v) + 1$ before the operation, and d is not changed by `Push`, we have $d(v) \leq d(u)$. Thus, d is legal after the operation. □

Lemma 2. *Let f be a preflow such that there exists a legal height function d with respect to f. Then, there exists no s-t-path in $\mathcal{N}_f$.*

Proof. Assume that such a path exists. Then, $d(s) = n$ and $d(t) = 0$, and $d(v) \geq d(u) - 1$ for every edge (u, v) on the path. But this is a contradiction, because the path can have at most $n - 1$ edges. □

Lemma 3. *If Algorithm 1 terminates, then f is a maximum flow in $\mathcal{N}$.*

Proof. The algorithm terminates if it cannot perform any `Push` or `Relabel` operation. This means that there are no active vertices, and thus, $e_f(v) = 0$ for all $v \in V \setminus \{s\}$. In particular, this means that f is a flow in $\mathcal{N}$, and since by Lemma 2 there are no s-t-paths in $\mathcal{N}_f$, we can conclude that f is a maximum flow in $\mathcal{N}$. □

It remains to analyze the running time of the algorithm.

Lemma 4. *Let f be a preflow, and let $w \in V$ be an active vertex with respect to f. Then, there exists a path in $\mathcal{N}_f$ from w to s.*

Proof. Let S be the set of vertices reachable from u in $\mathcal{N}_f$, and define $\bar{S} = V \setminus S$. Then,

$$\begin{aligned}
\sum_{v\in S} e_f(v) &= \sum_{v\in S}\left(\sum_{u\in V} f(u,v) - \sum_{u\in V} f(v,u)\right)\\
&= \sum_{v\in S}\left(\sum_{u\in S} f(u,v) - \sum_{u\in S} f(v,u)\right) + \sum_{v\in S}\left(\sum_{u\in \bar{S}} f(u,v) - \sum_{u\in \bar{S}} f(v,u)\right)\\
&= \sum_{v\in S}\left(\sum_{u\in \bar{S}} f(u,v) - \sum_{u\in \bar{S}} f(v,u)\right) \leq 0 \ ,
\end{aligned}$$

where the last inequality follows from the fact that for all $v \in S$ and all $u \in \bar{S}$, $f(u, v) = 0$ because $(v, u) \notin E_f$. Thus, there must be at least one negative term in $\sum_{v\in S} e_f(v)$, because by assumption of the lemma there exists $w \in S$ with $e_f(w) > 0$. Since f is a preflow, the only negative term can be $e_f(s)$. Thus, $s \in S$. □

Lemma 5. *For all $u \in V$, we always have $d(u) \leq 2n - 1$.*

Proof. Since the algorithm changes only the height of active vertices, it suffices to show that the height of an active vertex u is at most $2n - 1$. By Lemma 4, there is a path from u to s in $\mathcal{N}_f$. Since for each edge (w, v) on this path, $d(w) \leq d(v)+1$, we have $d(u) \leq d(s)+n-1 = 2n-1$. □

Corollary 1. *The algorithm performs at most $2n^2$ `Relabel` operations.*

We call a `Push(`u, v`)` *saturating* if it saturates the edge $(u, v) \in E_f$, i.e., if $c_f(u, v) = 0$ after the operation. Otherwise, we call it a *non-saturating* `Push` operation.

Lemma 6. *There are at most $2nm$ saturating* `Push` *operations.*

Proof. Consider an edge $(u, v) \in E \cup \overleftarrow{E}$. Between two saturating `Push` operations on (u, v), there must be a `Push` over (v, u). Since we only `Push` over (u, v) if $d(v) = d(u) - 1$ and over (v, u) if $d(u) = d(v) - 1$, we conclude that the height of u must increase by at least two between the two saturating `Push` operations over (u, v). By Lemma 5, the height of u can be at most $2n - 1$. Thus, there can be at most n saturating `Push` operations over (u, v), which proves the lemma. □

Lemma 7. *There are at most $6n^2m$ non-saturating* `Push` *operations.*

Proof. We first define a potential function Φ, which has a certain value at any time during the execution of the algorithm. Let A be the set of active vertices (at some point in time). We define

$$\Phi = \sum_{u \in A} d(u) \ .$$

We will now argue how the potential changes during the execution of the algorithm. Consider a non-saturating `Push` over (u, v). Thus, $e_f(u) < c_f(u, v)$, and by the definition of `Push` and the definition of $e_f(u)$, we have $e_f(u) = 0$ after `Push`(u, v). Therefore, u becomes inactive after `Push`(u, v), whereas v may or may not become active. Since $d(v) = d(u) - 1$, Φ decreases in any case by at least 1. A relabeling increases Φ by 1. A saturating `Push` makes at most one inactive vertex active, and thus, increases Φ by at most $2n - 1$ (by Lemma 5). Using the bounds of Lemma 6 and Corollary 1, the total increase of Φ over the whole execution of the algorithm is at most

$$2n^2 + 2nm(2n - 1) \leq 6n^2m \ .$$

Initially, s is inactive and $d(u) = 0$ for all over vertices, hence $\Phi = 0$. Moreover, by definition $\Phi \geq 0$. Thus, the total decrease of Φ cannot exceed its total increase. Since each non-saturating `Push` causes a decrease of at least 1, and the total increase is at most $6n^2m$, we conclude that there can be at most $6n^2m$ non-saturating `Push` operations. □

Since each `Init`, `Push`, and `Relabel` operation takes $O(1)$ time, we can conclude the following theorem.

Theorem 1. *The generic Preflow-Push algorithm computes a maximum flow in an s-t-network with n vertices and m edges in $O(n^2m)$ time.*

2 The Max-Height Preflow-Push Algorithm

The generic Preflow-Push algorithm leaves open in which order we perform operations when they are available. We will now discuss a specific implementation of Preflow-Push, the *Max-Height* algorithm, and given an improved running time for it. In particular, we will show an improved bound on the number of non-saturating `Push` operations, which is the main bottleneck in the bound of Theorem 1. As the name suggests, the Max-Height algorithm always chooses an active vertex with maximum height, and tries to push as much execess as possible from it to its neighbors. This subroutine is described below:

Function `Discharge`(u):
  while *u is active and at least one edge $(u, v) \in E_f$ is eligible* **do**
    `Push`(u,v)
  if *u is active* **then**
    `Relabel`(u)

We can now state the Max-Height algorithm as given in Algorithm 2.

Algorithm 2: The Max-Height Algorithm

Input: An s-t-network $\mathcal{N} = (V, E, c, s, t)$
`Init`($\mathcal{N}$)
while *at least one vertex is active* **do**
  $u \leftarrow$ an active vertex with height $d(u) = \max_{v \in A} d(v)$
  `Discharge`(u)
return f

To analyze the algorithm, we will again use a potential function. For a fixed state of the algorithm, let $d^* = \max_{v \in A} d(v)$ denote the maximum height of an active vertex (recall that A is the set of active vertices), and a^* denote the number of active vertices with height d^*. Moreover, for every $v \in V$, we denote $b(v) = |\{u \in V \mid d(u) \leq d(v)\}|$. We define a potential function $\Phi = \Phi_1 + \Phi_2 + \Phi_3$, where

- $\Phi_1 = Kd^*$,
- $\Phi_2 = \min\{a^*, K\}$, and
- $\Phi_3 = \frac{1}{K} \sum_{v \in V} b(v)$,

where $K \geq 1$ is a parameter that we will choose later. We write $\Delta\Phi_i$ to denote the change (increase) of Φ_i for some considered operation, for $i \in \{1, 2, 3\}$, and $\Delta\Phi := \Delta\Phi_1 + \Delta\Phi_2 + \Delta\Phi_3$.

Lemma 8. *For every* `Push` *operation, we have $\Delta\Phi_1 + \Delta\Phi_2 \leq 0$. Moreover, if d^* changes due to the* `Push`*, we have $\Delta\Phi_1 + \Delta\Phi_2 \leq -1$.*

Proof. If d^* does not change, clearly $\Delta\Phi_1 = 0$ and $\Delta\Phi_2 \leq 0$, because a `Push` from u to v is only possible if $d(u) = d(v)+1$, and thus, v does not contribute to a^*, but u may become inactive. For the same reason, no `Push` can increase d^*. If a `Push` decreases d^*, we have $\Delta\Phi_2 \leq K - 1$ (since $a^* \geq 1$ during the execution of the algorithm) and $\Delta\Phi_1 \leq -K$, yielding the stated bound. □

Lemma 9. *We have the following bounds on $\Delta\Phi$:*

- *If* `Relabel(`u`)` *is being executed, then $\Delta\Phi \leq K + n/K$.*
- *If* `Push(`u, v`)` *is being executed and saturating, then $\Delta\Phi \leq n/K$.*
- *If* `Push(`u, v`)` *is being executed and non-saturating, then $\Delta\Phi \leq -1$.*

Proof. If u is being relabelled, then $\Delta\Phi_1 \leq K$, $\Delta\Phi_2 \leq 0$, and $\Delta\Phi_3 \leq n/K$. The latter two bounds follow from the fact that A remains unchanged and $b(v)$ does not increase for any $v \neq u$.

If `Push` (u, v) is saturating, all b values are unchanged, and at most one more vertex becomes active. Together with Lemma 8, we conclude $\Delta\Phi \leq n/K$.

If `Push` (u, v) is non-saturating, all b values are unchanged, u becomes inactive, and v may become active. Hence, $\Delta\Phi_3 \leq (-b(u) + b(v))/K \leq -a/K < 0$, where a is value of a^* before the operation. If the `Push` changes d^*, this, together with Lemma 8, yields $\Delta\Phi \leq -1$. The same bound holds if d^* does not change, because then $\Delta\Phi_1 = 0$, $\Delta\Phi_2 \leq 0$, and either $a \geq K$ and hence $\Delta\Phi_3 \leq -1$, or $a < K$ and hence $\Delta\Phi_2 = -1$. In total, $\Delta\Phi \leq -1$. □

Lemma 10. *The number of non-saturating* `Push` *operations is at most $O(n^2\sqrt{m})$.*

Proof. We assume without loss of generality that $m \geq n - 1$. From Corollary 1 and Lemma 6, we know that there are at most $O(n^2)$ `Relabel` operations and at most $O(nm)$ saturating `Push` operations. Therefore, the total increase of Φ is $O(n^2(K+n/K)+nm\cdot n/K) = O(n^2K+n^2m/K)$. Since the initial value of Φ is $O(K+n^2/K)$, the total decrease in Φ is $O(n^2K+n^2m/K)$. Every non-saturating `Push` operation decreases Φ by at least 1, and since $\Phi \geq 0$, the number of non-saturating `Push` operations is at most $O(n^2K + n^2m/K)$. The lemma follows by choosing $K = \sqrt{m}$. □

Corollary 1, Lemma 6, and Lemma 10 show that the total number of performed operations of the Max-Height algorithm is at most $O(n^2\sqrt{m})$. We note here that it is also possible to implement the algorithm in a way that the total running time has this magnitude. In particular, keeping track of the active vertices of maximum height can be done using a clever data structure.

Theorem 2. *The Max-Height algorithm computes a maximum flow in an s-t-network with n vertices and m edges in $O(n^2\sqrt{m})$ time.*