Preflows

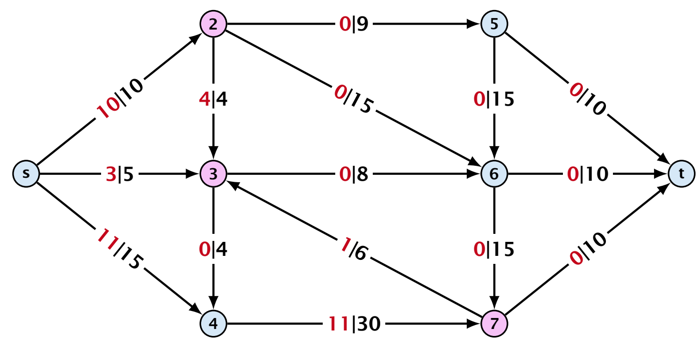
Definition 1
An (s,t)-preflow is a function $f : E \mapsto \mathbb{R}^+$ that satisfies

1. For each edge e
$$0 \le f(e) \le c(e) \ .$$
(capacity constraints)

2. For each $v \in V \setminus \{s,t\}$
$$\sum_{e \in \text{out}(v)} f(e) \le \sum_{e \in \text{into}(v)} f(e) \ .$$

Preflows

Example 2



A node that has $\sum_{e \in \text{out}(v)} f(e) < \sum_{e \in \text{into}(v)} f(e)$ is called an active node.

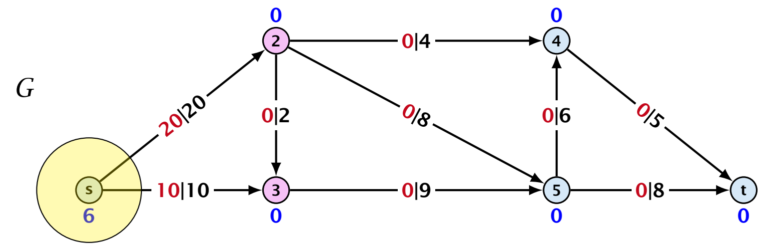
Preflows

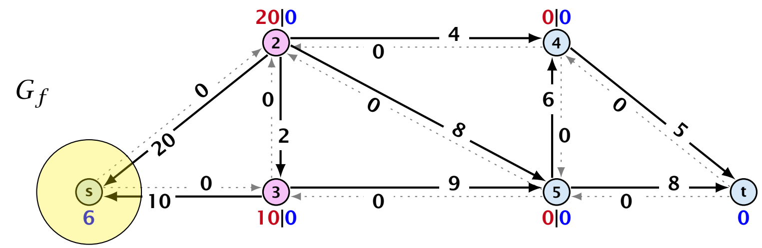
Definition:
A labelling is a function $\ell : V \to \mathbb{N}$. It is valid for preflow f if

- $\ell(u) \le \ell(v) + 1$ for all edges (u,v) in the residual graph G_f (only non-zero capacity edges!!!)
- $\ell(s) = n$
- $\ell(t) = 0$

Intuition:
The labelling can be viewed as a height function. Whenever the height from node u to node v decreases by more than 1 (i.e., it goes very steep downhill from u to v), the corresponding edge must be saturated.

Preflows





Preflows

Lemma 3
A preflow that has a valid labelling saturates a cut.

Proof:

- There are n nodes but $n+1$ different labels from $0, \dots, n$.
- There must exist a label $d \in \{0, \dots, n\}$ such that none of the nodes carries this label.
- Let $A = \{v \in V \mid \ell(v) > d\}$ and $B = \{v \in V \mid \ell(v) < d\}$.
- We have $s \in A$ and $t \in B$ and there is no edge from A to B in the residual graph G_f; this means that (A, B) is a saturated cut.

Lemma 4
A flow that has a valid labelling is a maximum flow.

Push Relabel Algorithms

Idea:

- start with some preflow and some valid labelling
- successively change the preflow while maintaining a valid labelling
- stop when you have a flow (i.e., no more active nodes)

Note that this is somewhat dual to an augmenting path algorithm. The former maintains the property that it has a feasible flow. It successively changes this flow until it saturates some cut in which case we conclude that the flow is maximum. A preflow push algorithm maintains the property that it has a saturated cut. The preflow is changed iteratively until it fulfills conservation constraints in which case we can conclude that we have a maximum flow.

Changing a Preflow

An arc (u, v) with $c_f(u, v) > 0$ in the residual graph is admissible if $\ell(u) = \ell(v) + 1$ (i.e., it goes downwards w.r.t. labelling ℓ).

The push operation
Consider an active node u with excess flow $f(u) = \sum_{e \in \mathrm{into}(u)} f(e) - \sum_{e \in \mathrm{out}(u)} f(e)$ and suppose $e = (u, v)$ is an admissible arc with residual capacity $c_f(e)$.

We can send flow $\min\{c_f(e), f(u)\}$ along e and obtain a new preflow. The old labelling is still valid (!!!).

- saturating push: $\min\{f(u), c_f(e)\} = c_f(e)$
  the arc e is deleted from the residual graph
- non-saturating push: $\min\{f(u), c_f(e)\} = f(u)$
  the node u becomes inactive

Note that a push-operation may be saturating **and** non-saturating at the same time.

Push Relabel Algorithms

The relabel operation
Consider an active node u that does not have an outgoing admissible arc.

Increasing the label of u by 1 results in a valid labelling.

- Edges (w, u) incoming to u still fulfill their constraint $\ell(w) \le \ell(u) + 1$.
- An outgoing edge (u, w) had $\ell(u) < \ell(w) + 1$ before since it was not admissible. Now: $\ell(u) \le \ell(w) + 1$.

Push Relabel Algorithms

Intuition:
We want to send flow downwards, since the source has a height/label of n and the target a height/label of 0. If we see an active node u with an admissible arc we push the flow at u towards the other end-point that has a lower height/label. If we do not have an admissible arc but excess flow into u it should roughly mean that the level/height/label of u should rise. (If we consider the flow to be water then this would be natural.)

Note that the above intuition is very incorrect as the labels are integral, i.e., they cannot really be seen as the height of a node.

Reminder

- In a preflow nodes may not fulfill conservation constraints; a node may have more incoming flow than outgoing flow.
- Such a node is called active.
- A labelling is valid if for every edge (u, v) in the residual graph $\ell(u) \leq \ell(v) + 1$.
- An arc (u, v) in residual graph is admissible if $\ell(u) = \ell(v) + 1$.
- A saturating push along e pushes an amount of $c(e)$ flow along the edge, thereby saturating the edge (and making it dissappear from the residual graph).
- A non-saturating push along $e = (u, v)$ pushes a flow of $f(u)$, where $f(u)$ is the excess flow of u. This makes u inactive.

Push Relabel Algorithms

Algorithm 3 maxflow(G, s, t, c)

find initial preflow f
while there is active node u **do**
  if there is admiss. arc e out of u **then**
    push(G, e, f, c)
  else
    relabel(u)
return f

In the following example we always stick to the same active node u until it becomes inactive but this is not required.

Preflow Push Algorithm

Animation for push relabel algorithms is only available in the lecture version of the slides.

Analysis

Note that the lemma is almost trivial. A node v having excess flow means that the current preflow ships something to v. The residual graph allows to *undo* flow. Therefore, there must exist a path that can undo the shipment and move it back to s. However, a formal proof is required.

Lemma 5
An active node has a path to s in the residual graph.

Proof.

- Let A denote the set of nodes that can reach s, and let B denote the remaining nodes. Note that $s \in A$.
- In the following we show that a node $b \in B$ has excess flow $f(b) = 0$ which gives the lemma.
- In the residual graph there are no edges into A, and, hence, no edges leaving A/entering B can carry any flow.
- Let $f(B) = \sum_{v \in B} f(v)$ be the excess flow of all nodes in B.

Let $f : E \to \mathbb{R}_0^+$ be a preflow. We introduce the notation

$$f(x,y) = \begin{cases} 0 & (x,y) \notin E \\ f((x,y)) & (x,y) \in E \end{cases}$$

We have

$$\begin{aligned} f(B) &= \sum_{b \in B} f(b) \\ &= \sum_{b \in B} \Big(\sum_{v \in V} f(v,b) - \sum_{v \in V} f(b,v) \Big) \\ &= \sum_{b \in B} \Big(\sum_{v \in A} f(v,b) + \sum_{v \in B} f(v,b) - \sum_{v \in A} f(b,v) - \sum_{v \in B} f(b,v) \Big) \\ &= \qquad - \sum_{b \in B} \sum_{v \in A} f(b,v) \\ &\le 0 \end{aligned}$$

Hence, the excess flow $f(b)$ must be 0 for every node $b \in B$.

Analysis

Lemma 6
The label of a node cannot become larger than $2n - 1$.

Proof.

- When increasing the label at a node u there exists a path from u to s of length at most $n - 1$. Along each edge of the path the height/label can at most drop by 1, and the label of the source is n.

Lemma 7
There are only $\mathcal{O}(n^2)$ relabel operations.

Analysis

Lemma 8
The number of saturating pushes performed is at most $\mathcal{O}(mn)$.

Proof.

- Suppose that we just made a saturating push along (u,v).
- Hence, the edge (u,v) is deleted from the residual graph.
- For the edge to appear again, a push from v to u is required.
- Currently, $\ell(u) = \ell(v) + 1$, as we only make pushes along admissible edges.
- For a push from v to u the edge (v,u) must become admissible. The label of v must increase by at least 2.
- Since the label of v is at most $2n - 1$, there are at most n pushes along (u,v).

Lemma 9
The number of non-saturating pushes performed is at most $\mathcal{O}(n^2m)$.

Proof.

- Define a potential function $\Phi(f) = \sum_{\text{active nodes} v} \ell(v)$
- A saturating push increases Φ by $\leq 2n$ (when the target node becomes active it may contribute at most $2n$ to the sum).
- A relabel increases Φ by at most 1.
- A non-saturating push decreases Φ by at least 1 as the node that is pushed from becomes inactive and has a label that is strictly larger than the target.
- Hence,

$$\begin{aligned}\#\text{non-saturating\_pushes} &\leq \#\text{relabels} + 2n \cdot \#\text{saturating\_pushes}\\ &\leq \mathcal{O}(n^2m)\ .\end{aligned}$$

Analysis

Theorem 10
There is an implementation of the generic push relabel algorithm with running time $\mathcal{O}(n^2m)$.

Analysis

Proof:

For every node maintain a list of admissible edges starting at that node. Further maintain a list of active nodes.

A push along an edge (u,v) can be performed in constant time

- check whether edge (v,u) needs to be added to G_f
- check whether (u,v) needs to be deleted (saturating push)
- check whether u becomes inactive and has to be deleted from the set of active nodes

A relabel at a node u can be performed in time $\mathcal{O}(n)$

- check for all outgoing edges if they become admissible
- check for all incoming edges if they become non-admissible

Analysis

For special variants of push relabel algorithms we organize the neighbours of a node into a linked list (possible neighbours in the residual graph G_f). Then we use the discharge-operation:

Algorithm 4 discharge(u)

while u is active **do**
    $v \leftarrow u.\textit{current-neighbour}$
    if $v = \text{null}$ **then**
        relabel(u)
        $u.\textit{current-neighbour} \leftarrow u.\textit{neighbour-list-head}$
    else
        if (u,v) admissible **then** push(u,v)
        else $u.\textit{current-neighbour} \leftarrow v.\textit{next-in-list}$

Note that $u.\textit{current-neighbour}$ is a global variable. It is only changed within the discharge routine, but keeps its value between consecutive calls to discharge.

In order for e to become admissible the other end-point say v has to push flow to u (so that the edge (u, v) re-appears in the residual graph). For this the label of v needs to be larger than the label of u. Then in order to make (u, v) admissible the label of u has to increase.

Lemma 11
If $v = \text{null}$ in Line 3, then there is no outgoing admissible edge from u.

Proof.

- While pushing from u the current-neighbour pointer is only advanced if the current edge is not admissible.
- The only thing that could make the edge admissible again would be a relabel at u.
- If we reach the end of the list ($v = \text{null}$) all edges are not admissible. □

This shows that discharge(u) is correct, and that we can perform a relabel in Line 4.

13.2 Relabel to Front

Algorithm 21 relabel-to-front(G, s, t)

1: initialize preflow
2: initialize node list L containing $V \setminus \{s, t\}$ in any order
3: **foreach** $u \in V \setminus \{s, t\}$ **do**
4: $\quad u.\textit{current-neighbour} \leftarrow u.\textit{neighbour-list-head}$
5: $u \leftarrow L.\textit{head}$
6: **while** $u \neq \text{null}$ **do**
7: $\quad \textit{old-height} \leftarrow \ell(u)$
8: $\quad$ discharge(u)
9: $\quad$ **if** $\ell(u) > \textit{old-height}$ **then** // relabel happened
10: $\quad\quad$ move u to the front of L
11: $\quad u \leftarrow u.\textit{next}$

13.2 Relabel to Front

Lemma 12 (Invariant)
In Line 6 of the relabel-to-front algorithm the following invariant holds.

1. *The sequence L is topologically sorted w.r.t. the set of admissible edges; this means for an admissible edge (x, y) the node x appears before y in sequence L.*
2. *No node before u in the list L is active.*

Proof:

- Initialization:
  1. In the beginning s has label $n \geq 2$, and all other nodes have label 0. Hence, no edge is admissible, which means that any ordering L is permitted.
  2. We start with u being the head of the list; hence no node before u can be active
- Maintenance:
  1. - Pushes do no create any new admissible edges. Therefore, if discharge$()$ does not relabel u, L is still topologically sorted.
     - After relabeling, u cannot have admissible incoming edges as such an edge (x, u) would have had a difference $\ell(x) - \ell(u) \geq 2$ before the re-labeling (such edges do not exist in the residual graph).
       Hence, moving u to the front does not violate the sorting property for any edge; however it fixes this property for all admissible edges leaving u that were generated by the relabeling.

13.2 Relabel to Front

Proof:

- Maintenance:
  2. If we do a relabel there is nothing to prove because the only node before u' (u in the next iteration) will be the current u; the discharge(u) operation only terminates when u is not active anymore.

     For the case that we do not relabel, observe that the only way a predecessor could be active is that we push flow to it via an admissible arc. However, all admissible arc point to successors of u.

Note that the invariant means that for $u = \text{null}$ we have a preflow with a valid labelling that does not have active nodes. This means we have a maximum flow.

13.2 Relabel to Front

Lemma 13
There are at most $\mathcal{O}(n^3)$ calls to discharge(u).

Every discharge operation without a relabel advances u (the current node within list L). Hence, if we have n discharge operations without a relabel we have $u = \text{null}$ and the algorithm terminates.

Therefore, the number of calls to discharge is at most $n(\#relabels + 1) = \mathcal{O}(n^3)$.

13.2 Relabel to Front

Lemma 14
The cost for all relabel-operations is only $\mathcal{O}(n^2)$.

A relabel-operation at a node is constant time (increasing the label and resetting $u.current\text{-}neighbour$). In total we have $\mathcal{O}(n^2)$ relabel-operations.

13.2 Relabel to Front

Note that by definition a saturating push operation ($\min\{c_f(e), f(u)\} = c_f(e)$) can at the same time be a non-saturating push operation ($\min\{c_f(e), f(u)\} = f(u)$).

Lemma 15
The cost for all saturating push-operations that are **not** *also non-saturating push-operations is only $\mathcal{O}(mn)$.*

Note that such a push-operation leaves the node u active but makes the edge e disappear from the residual graph. Therefore the push-operation is immediately followed by an increase of the pointer $u.current\text{-}neighbour$.
This pointer can traverse the neighbour-list at most $\mathcal{O}(n)$ times (upper bound on number of relabels) and the neighbour-list has only $degree(u) + 1$ many entries ($+1$ for null-entry).

13.2 Relabel to Front

Lemma 16

The cost for all non-saturating push-operations is only $\mathcal{O}(n^3)$.

A non-saturating push-operation takes constant time and ends the current call to discharge(). Hence, there are only $\mathcal{O}(n^3)$ such operations.

Theorem 17

The push-relabel algorithm with the rule relabel-to-front takes time $\mathcal{O}(n^3)$.

13.3 Highest Label

Algorithm 6 highest-label(G, s, t)

1: initialize preflow
2: **foreach** $u \in V \setminus \{s, t\}$ **do**
3: $\quad u.current\text{-}neighbour \leftarrow u.neighbour\text{-}list\text{-}head$
4: **while** $\exists$ active node u **do**
5: $\quad$ select active node u with highest label
6: $\quad$ discharge(u)

13.3 Highest Label

Lemma 18

When using highest label the number of non-saturating pushes is only $\mathcal{O}(n^3)$.

A push from a node on level ℓ can only “activate” nodes on levels strictly less than ℓ.

This means, after a non-saturating push from u a relabel is required to make u active again.

Hence, after n non-saturating pushes without an intermediate relabel there are no active nodes left.

Therefore, the number of non-saturating pushes is at most $n(\#relabels + 1) = \mathcal{O}(n^3)$.

13.3 Highest Label

Since a discharge-operation is terminated by a non-saturating push this gives an upper bound of $\mathcal{O}(n^3)$ on the number of discharge-operations.

The cost for relabels and saturating pushes can be estimated in exactly the same way as in the case of the generic push-relabel algorithm.

Question:

How do we find the next node for a discharge operation?

13.3 Highest Label

Maintain lists L_i, $i \in \{0, \ldots, 2n\}$, where list L_i contains active nodes with label i (maintaining these lists induces only constant additional cost for every push-operation and for every relabel-operation).

After a discharge operation terminated for a node u with label k, traverse the lists $L_k, L_{k-1}, \ldots, L_0$, (in that order) until you find a non-empty list.

Unless the last (non-saturating) push was to s or t the list $k-1$ must be non-empty (i.e., the search takes constant time).

13.3 Highest Label

Hence, the total time required for searching for active nodes is at most

$$\mathcal{O}(n^3) + n(\textit{\#non-saturating-pushes-to-s-or-t})$$

Lemma 19

The number of non-saturating pushes to s or t is at most $\mathcal{O}(n^2)$.

With this lemma we get

Theorem 20

The push-relabel algorithm with the rule highest-label takes time $\mathcal{O}(n^3)$.

13.3 Highest Label

Proof of the Lemma.

- We only show that the number of pushes to the source is at most $\mathcal{O}(n^2)$. A similar argument holds for the target.
- After a node v (which must have $\ell(v) = n + 1$) made a non-saturating push to the source there needs to be another node whose label is increased from $\leq n + 1$ to $n + 2$ before v can become active again.
- This happens for every push that v makes to the source. Since, every node can pass the threshold $n + 2$ at most once, v can make at most n pushes to the source.
- As this holds for every node the total number of pushes to the source is at most $\mathcal{O}(n^2)$.