PUSH-RELABEL ALGORITHMS

CSCI 4113/6101
INSTRUCTOR: NORBERT ZEH
SEPTEMBER 23, 2025

One of the key characteristics of augmenting path algorithms is that they maintain a feasible flow at all times. In this topic, we discuss **push-relabel algorithms**, which use a completely different strategy to find a maximum flow. We will discuss the most basic push-relabel algorithm in this topic, which runs in $O\left(n^2m\right)$ time. Just as all augmenting path algorithms are variants of the Ford-Fulkerson algorithm that use better ways to find augmenting paths, the push-relabel algorithm we discuss in this topic will leave some choices the algorithm makes unspecified. By being careful about these choices, one can obtain significantly faster maximum flow algorithms than the one discussed here. By combining these strategies with efficient data structures to maintain the state of the algorithm, one obtains the fastest maximum flow algorithms known to date.

Consider the st-cut $S = \{s\}$. The capacity $U_{\{s\}}$ of this cut is the total capacity of the out-edges of s. By the Max-Flow Min-Cut Theorem, we cannot move more than $U_{\{s\}}$ units of flow from s to t. The idea of push-relabel algorithms is to be optimistic and to start by saturating the out-edges of s by moving $u_{s,x}$ flow along every out-edge (s, x) of s. This means that the out-neighbours of s no longer satisfy flow conservation; the current flow is not a feasible flow. To restore flow conservation of the out-neighbours of s, we move the flow they receive from s along *their* out-edges to *their* out-neighbours. Now other vertices violate flow conservation. We continue this game until some flow eventually reaches t. Since we were probably overly optimistic in sending $U_{\{s\}}$ units of flow from s to its out-neighbours, some of the flow that originated at s will never be able to reach t and ends up "sloshing around the network". Once we are sure that as much flow as possible has reached t, we send the flow that was not able to reach t back to s. This restores flow conservation everywhere and gives us a maximum flow.

This is the intuition behind how push-relabel algorithms work. There are a number of questions we need to answer about this algorithm: How do we choose the out-neighbours to which we move flow from a given vertex? Does the choice of these out-neighbours matter for how quickly we find a maximum flow? How do we decide that no more flow can reach t? How exactly do we send flow "back to s"? And many more. We will answer all these questions in time.

1 PREFLOWS, HEIGHT FUNCTIONS, PUSH, AND RELABEL

The name "push-relabel algorithms" stems from the two basic operations these algorithms use to find a maximum flow: PUSH and RELABEL. We will discuss them shortly. An alternative name by which these algorithms are knows is "preflow-push algorithms", because the function $f : V \to \mathbb{R}_0^+$ they maintain is a **preflow**. A preflow is a function $f : E \to \mathbb{R}_0^+$ such that

$$0 \leq f_e \leq u_e \qquad \forall e \in E$$

and

$$\sum_{y \in V} \left(f_{y,x} - f_{x,y} \right) \geq 0 \qquad \forall x \in V \setminus \{s, t\}.$$

In words, f satisfies the capacity constraints, and every vertiex in $V \setminus \{s, t\}$ either satisfies flow conservation or receives more flow from its in-neighbours than it sends to its out-neighbours. The quantity

$$e_x = \sum_{y \in V} \left(f_{y,x} - f_{x,y} \right) \geq 0$$

is called the **excess** at node x.[1] This means that if every vertex has excess 0, then f is a feasible flow.

As just explained, every push-relabel algorithm starts with the preflow

$$f_{x,y} = \begin{cases} u_{x,y} & \text{if } x = s \\ 0 & \text{if } x \neq s. \end{cases}$$

If some vertex x has positive excess, $e_x > 0$, and there exists an edge (x, y) with residual capacity $u^f_{x,y} > 0$, then we can move $\delta = \min\left(e_x, u^f_{x,y}\right)$ flow from x to y along this edge. This is called a **PUSH** operation. The choice of δ ensures that we maintain a preflow:

- By pushing no more than e_x flow along the edge (x, y), we ensure that the excess of x remains non-negative.
- By pushing no more than $u^f_{x,y}$ flow along the edge (x, y), we ensure that all edges have non-negative residual capacities, that is, that $0 \leq f_{x,y} \leq u_{x,y}$, for every edge (x, y) of G.

Note that after pushing flow along the edge (x, y), y has positive excess and $u^f_{y,x} > 0$. Thus, the next step could push the flow we just sent from x to y back to x, along the edge (y, x). If we did this, we would never make any progress towards finding a feasible flow. Thus, we need a tool that helps us choose along which edge to push flow in each iteration. The next definition provides us with this tool. A **height function** is a labelling $h : V \to \mathbb{N}$ that satisfies

$$\begin{aligned} h_s &= n, \\ h_t &= 0, \\ h_x &\leq h_y + 1 \quad \forall (x, y) \in G^f. \end{aligned}$$

The algorithm is allowed to push flow along the edge (x, y) only if

- $e_x > 0$,
- $u^f_{x,y} > 0$, and
- $h_x = h_y + 1$.

As the algorithm progresses, it may reach a state in which there exist vertices with positive excess and with out-edges in G^f along which we could push flow, only we are not allowed to push flow along those out-edges because each such edge (x, y) satisfies $h_x \leq h_y$. This is where the second operation comes into play. A **RELABEL** operation chooses a vertex x with $e_x > 0$ and updates its height to

$$h_x = 1 + \min\left\{ h_y \,\middle|\, (x, y) \in G^f \right\}.$$

[1] This is exactly the negation of the "deficit" d_x we used in the proof of the Max-Flow Min-Cut Theorem via LP duality.

A potentially helpful metaphor that builds some intuition but unfortunately provides little help with proving the correctness of the algorithm is the following: Assume that s initially holds $\sum_{y \in V} u_{s,y}$ units of flow (litres of water). We can make this water flow to s's out-neighbours by raising s so it is higher than its neighbours and the water flows downhill to s's neighbours. We raise s and its neighbours further to make the water flow onward to the vertices two hops away from s, and so on. At some point, t is the lowest vertex and some of the water reaches t while some water may be stuck at intermediate vertices (because their out-edges are at capacity). By lifting up vertices so they are higher than s, we make this excess water flow back to s. RELABEL operations implement this lifting up of vertices. PUSH operations implement the downward flow of water along edges in G^f.

2 THE PUSH-RELABEL ALGORITHM

We have all the tools we need to describe the push-relabel algorithm now:

- The algorithm starts by intializing the preflow and height function to

$$f_{x,y} = \begin{cases} u_{x,y} & \text{if } x = s \\ 0 & \text{if } x \neq s. \end{cases}$$

$$h_x = \begin{cases} n & \text{if } x = s \\ 0 & \text{if } x \neq s. \end{cases}$$

- PUSH: If there exists an edges $(x, y) \in G^f$ with $e_x > 0$ and $h_x = h_y + 1$, then it may push $\delta = \min\left(e_x, u^f_{x,y}\right)$ flow along this edge.
- RELABEL: If there exists a vertex x with $e_x > 0$ and $h_x \leq h_y$, for every out-neighbour y of x in G^f, then it may change the height of x to $h_x = 1 + \min\left\{h_y \,\middle|\, (x, y) \in G^f\right\}$.
- The algorithm performs these PUSH and RELABEL operations until f is a feasible flow. When this happens, the algorithm terminates and returns f.

Fig. 1 illustrates a run of the push-relabel algorithm.

Just as we did not specify for the Ford-Fulkerson algorithm how to choose the augmenting path in each iteration, this description of the push-relabel algorithm does not specify how to choose between PUSH and RELABEL operations nor how to choose the vertex to relabel or the edge along which to push flow. In the case of the Fold-Fulkerson algorithm, a careful choice of the augmenting path was necessary to ensure termination of the algorithm. In contrast, even choosing completely arbitrarily between PUSH and RELABEL operations and between the edges or vertices to which to apply these operations (subject to the stated conditions under which these operations *can* be applied to an edge or vertex) results in an algorithm that runs $O\left(n^2 m\right)$ time. More careful, and sometimes somewhat unintuitive, choices lead to push-relabel algorithms with running times $O\left(n^3\right)$ and $O\left(n^2\sqrt{m}\right)$. Even faster push-relabel algorithms can be obtained by using efficient data structures called link-cut trees.

The remainder of this topic proves that the push-relabel algorithm is correct and that it does indeed run in $O\left(n^2 m\right)$ time.

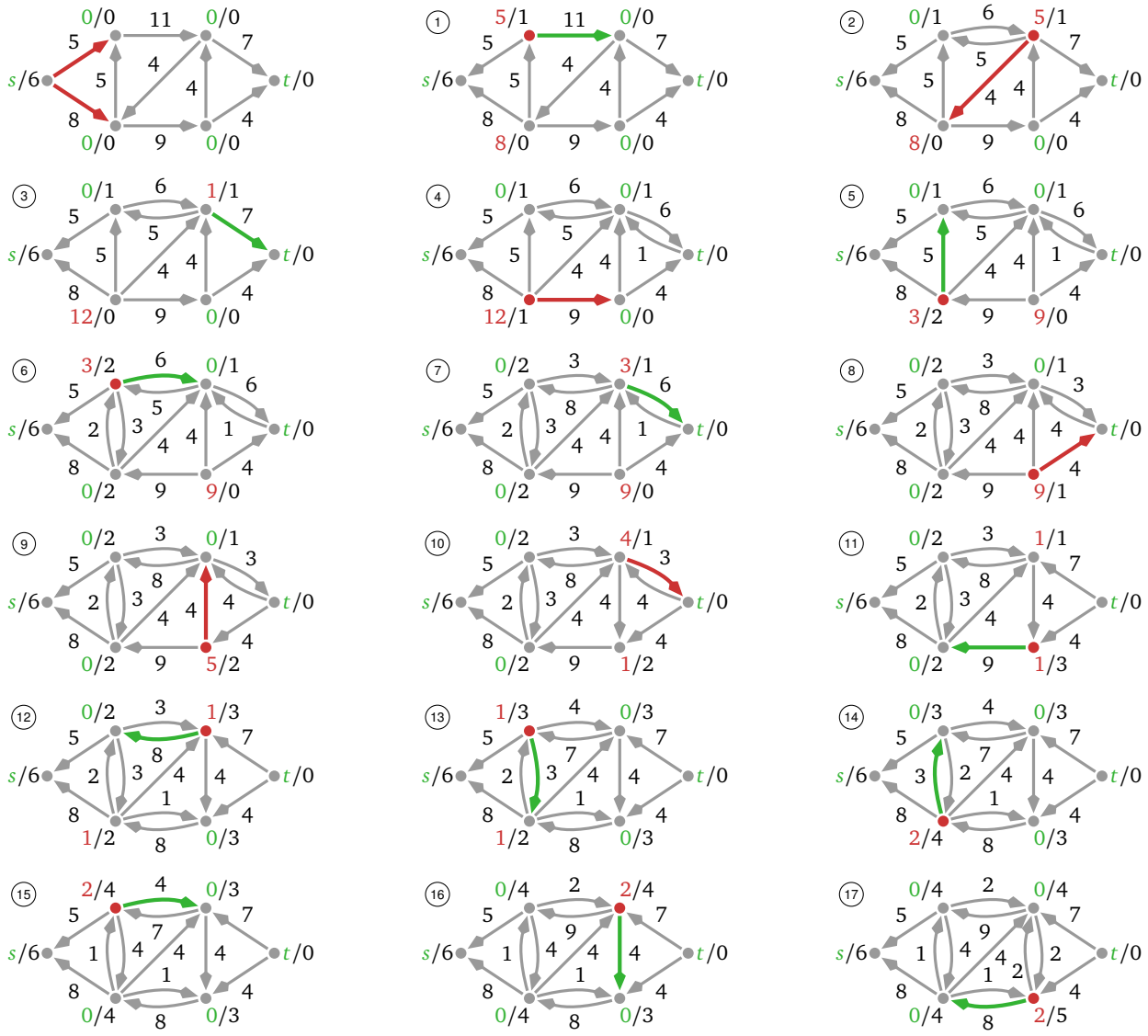

Figure 1: Illustration of an application of the push-relabel algorithm to the network in the top-left figure. The top-left figure shows the initialization of the flow and height function. For the remaining figures, the circled numbers indicate the iteration number. Only the residual network is shown. Edge labels are edge capacities. Black node labels are node heights. Coloured node labels are node excesses, shown in red if positive and in green if zero. s and t are labelled with their names instead, to make them recognizable; they never have any excess. In each figure, the edge used to push flow to a neighbour is shown in red if it is a saturating push or in green if it is a non-saturating push. If the tail of the edge was relabelled to allow this push to happen, this tail node is shown in red. The final network in the bottom row on the next page shows the computed flow. (Continued on next page.)

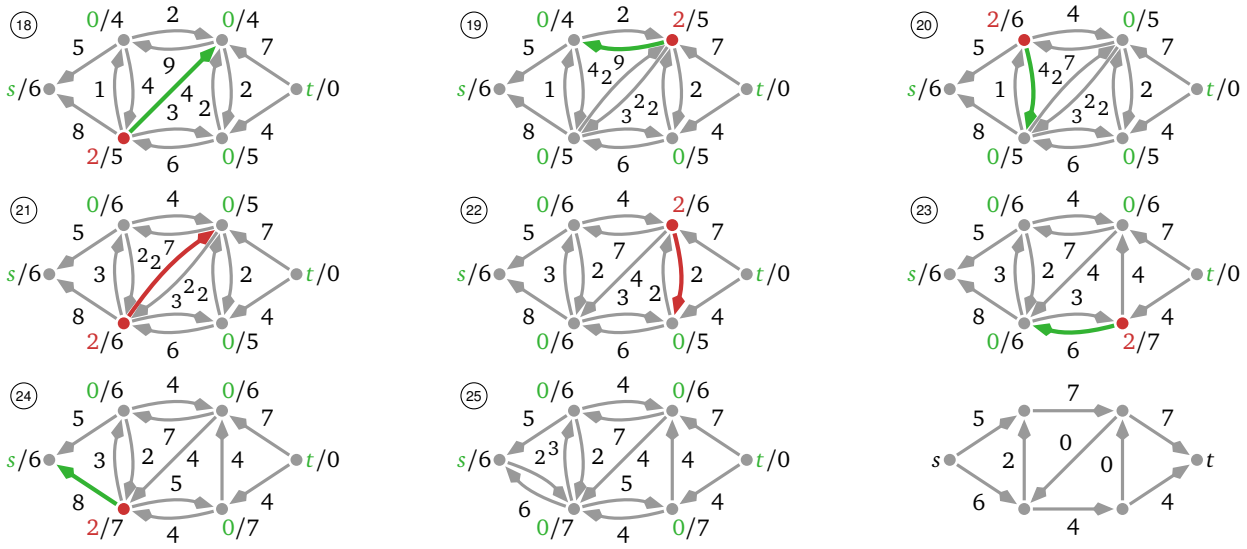

Figure 1: Continued from previous page

3 Correctness

We will prove in the next section that the push-relabel algorithm terminates, and that it does so quickly. Thus, it does return an edge labelling f. To establish the correctness of the algorithm, we need to prove that f is a maximum flow at that time. To this end, we prove that the initial flow is a preflow and that the initial vertex labelling h is a valid height function with respect to f. Then we prove that PUSH and RELABEL operations maintain that f is a preflow and h is a valid height function with respect to f. Once the algorithm terminates, there is no vertex with positive excess. Thus, f satisfies flow conservation at this time and, therefore, is a feasible flow. This proves that the algorithm returns a feasible flow. To prove that it is a *maximum* flow, we will use the properties of the height function to prove that G^f contains no augmenting path. As shown in the correctness proof of the Ford-Fulkerson algorithm, this implies that there exists an st-cut S with $F_s = U_S$. By the Max-Flow Min-Cut Theorem, this shows that f is a maximum flow.

3.1 Maintenance of Preflow and Height Function

Lemma 1. *The initial edge labelling*

$$f_{x,y} = \begin{cases} u_{x,y} & \text{if } x = s \\ 0 & \text{if } x \neq s \end{cases}$$

is a preflow, and the initial vertex labelling

$$h_x = \begin{cases} n & \text{if } x = s \\ 0 & \text{if } x \neq s \end{cases}$$

is a valid height function with respect to f.

Proof. The definition of f explicitly ensures that $0 \leq f_{x,y} \leq u_{x,y}$, for every edge $(x,y) \in G$. No vertex $x \neq s$ has an out-edge (x,y) with $f_{x,y} > 0$. Thus, every vertex $x \in V \setminus \{s,t\}$ has non-negative excess. This shows that f is a preflow.

For all $y \in V$, we have $f_{s,y} = u_{s,y}$ and $f_{y,s} = 0$. Thus, $u^f_{s,y} = u_{s,y} - f_{s,y} + f_{y,s} = 0$. This shows that G^f does not contain any out-edges of s. For every edge $(x,y) \in G^f$ with $x \neq s$, we have $h_x = 0$ and $h_y \geq 0$. Thus, $h_x \leq h_y + 1$ for each such edge. This shows that h is a valid height function with respect to f. □

LEMMA 2. *If f is a preflow and h is a valid height function with respect to f before a* PUSH *operation, then the same is true after this* PUSH *operation.*

Proof. Pushing $\delta = \min\left(e_x, u^f_{x,y}\right)$ units of flow along the edge (x,y) constructs a new flow f' with

$$f'_{v,w} = \begin{cases} f_{v,w} - \min(\delta, f_{v,w}) & \text{if } (v,w) = (y,x) \\ f_{v,w} + \max(0, \delta - f_{w,v}) & \text{if } (v,w) = (x,y) \\ f_{v,w} & \text{otherwise.} \end{cases}$$

To prove that f' is a preflow, we need to show that it satisfies flow conservation and that every vertex in $V \setminus \{s,t\}$ has non-negative excess with respect to f'.

For every edge $(v,w) \notin \{(x,y),(y,x)\}$, we have $0 \leq f'_{v,w} \leq u_{v,w}$ because $f'_{v,w} = f_{v,w}$ and $0 \leq f_{v,w} \leq u_{v,w}$.

For the edges (x,y) and (y,x), we have

$$0 \leq f_{y,x} - \min(\delta, f_{y,x}) \leq f_{y,x} \leq u_{y,x}$$

and

$$\begin{aligned} 0 &\leq f_{x,y} \\ &\leq f_{x,y} + \max(0, \delta - f_{y,x}) \\ &\leq f_{x,y} + \max(0, u^f_{x,y} - f_{y,x}) \\ &= f_{x,y} + \max(0, u_{x,y} - f_{x,y} + f_{y,x} - f_{y,x}) \\ &= \max(f_{x,y}, u_{x,y}) \\ &= u_{x,y}. \end{aligned}$$

Thus, since $f'_{x,y} = f_{x,y} + \max(0, \delta - f_{y,x})$ an $f'_{y,x} = f_{y,x} - \min(\delta, f_{y,x})$, we have $0 \leq f'_{x,y} \leq u_{x,y}$ and $0 \leq f'_{y,x} \leq u_{y,x}$. This shows that f' satisifies flow conservation.

Let e'_v be the excess of v with respect to f'. Then we have $e'_v \geq e_v \geq 0$, for all $v \neq x$. For $v = x$, we have $e'_x = e_x - \delta = e_x - \min\left(e_x, u^f_{x,y}\right) \geq 0$. Thus, $e'_v \geq 0$, for all $v \in V \setminus \{s,t\}$.

A PUSH operation does not change the height function. Thus, since h is a valid height function with respect to f, it is a valid height function with respect to f' unless $G^{f'}$ contains an edge (v,w) not in G^f with $h_v > h_w + 1$. However, pushing flow along the edge (x,y) can only add the edge (y,x) as a new edge of $G^{f'}$ that is not an edge of G^f (provided $(y,x) \notin G^f$). Since we push flow along the edge (x,y), h satisfies $h_x = h_y + 1$ and, therefore, $h_y \leq h_x + 1$. Thus, h is a valid height function with respect to f'. □

LEMMA 3. *Relabelling a vertex x strictly increases h_x.*

Proof. We relabel x only if $h_x \leq \min\left\{h_y \,\middle|\, (x,y) \in G^f\right\}$. The label of x after relabelling is $h'_x = 1 + \min\left\{h_y \,\middle|\, (x,y) \in G^f\right\} \geq h_x + 1$. □

LEMMA 4. *If f is a preflow and h is a valid height function with respect to f before a* RELABEL *operation, then the same is true after this* RELABEL *operation.*

Proof. Since a RELABEL operation does not change f, f clearly remains a preflow. Let h' be the new height function after relabelling a vertex x. Then $h'_v = h_v$, for all $v \neq x$, and, by Lem. 3, $h'_x \geq h_x + 1$. Thus, for every edge (v,w) with $v \neq x$, we have $h'_v = h_v \leq h_w + 1 \leq h'_w + 1$. If $v = x$, then $h'_x = 1 + \min\left\{h_y \,\middle|\, (x,y) \in G^f\right\} \leq 1 + h_w = 1 + h'_w$. Thus, h' is a valid height function with respect to f. □

Lems. 1, 2 and 4 together prove that at all times during the execution of the algorithm, f is a preflow, and h is a valid height function with respect to f. As alrealy observed, this implies that

PROPOSITION 5. *The edge labelling f returned by the push-relabel algorithm is a feasible flow.*

To ensure that the algorithm at least has a chance to terminate, we should verify that it never gets "stuck". More precisely, we need that

LEMMA 6. *As long as there exists a vertex with positive excess, there exists an edge (x,y) to which a* PUSH *operation can be applied or a vertex to which a* RELABEL *operation can be applied.*

Proof. By Lems. 1, 2 and 4, h is a valid height function with respect to f at all times. Thus, for every edge $(x,y) \in G^f$ we have $h_x \leq h_y + 1$. If x has positive excess and $h_x = h_y + 1$, then we can push flow along the edge (x,y). If $h_x \leq h_y$, for all out-edges (x,y) of x in G^f, then x can be relabelled. □

3.2 THE RETURNED FLOW IS A MAXIMUM FLOW

By Prop. 5, the edge labelling f returned by the push-relabel algorithm is a feasible flow. Next we prove that at no time during the execution of the algorithm, does there exist a path from s to t in G^f. Thus, once f becomes feasible, the vertices reachable from s in G^f form an st-cut S with with $F_s = U_S$, as shown in the correctness proof of the Ford-Fulkerson algorithm. Therefore, by the Max-Flow Min-Cut Theorem, f is a maximum flow.

PROPOSITION 7. *The flow returned by the push-relabel algorithm is a maximum flow.*

Proof. As just discussed, we need to prove that there is no path from s to t in G^f. Assume there exists such a path $P = \langle s = x_0, \ldots, x_k = t \rangle$. Then $k \leq n-1$. Since h is a valid height function with respect to t, we have $h_{x_i} \leq h_{x_{i+1}} + 1$, for all $0 \leq i < k$, and, therefore, $h_s \leq h_t + k < h_t + n$. However, a valid height function h also satisfies $h_s = n$ and $h_t = 0$. Thus, $h_s = h_t + n$, a contradiction. □

4 RUNNING TIME

Having established the correctness of the push-relabel algorithm, it remains to bound its running time. We start by bounding the costs of individual PUSH and RELABEL operations. Then we count how many operation of each type the push-relabel algorithm performs.

4.1 The Costs of Individual Operations

For the push-relabel algorithm to be efficient, we need to use a simple data structure to represent the network G and the residual network G^f. The undirected versions of these two networks are subgraphs of the graph $G' = (V, E')$, where $E' = \{\{x, y\} \mid (x, y) \in G \vee (y, x) \in G\}$. We store G' in a slightly modified adjacency list representation. To represent G, every edge $\{x, y\}$ of G' stores whether G contains the edge (x, y), (y, x) or both. For each of these edges e, it stores its capacity u_e and the flow f_e along it. Note that this allows us to compute in constant time which of the two edges (x, y) and (y, x) belong to G^f, along with their residual capacities. Every vertex x of G' stores its height h_x and its excess e_x. We split the vertex list of G' into two sublists, the first storing all vertices with positive excess, the second storing all vertices with excess 0. Similarly, we split the adjacency list of every vertex x into two sublists. The first sublist stores all "pushable" edges $\{x, y\}$ incident to x. These are edges $\{x, y\}$ such that $(x, y) \in G^f$ and $h_x = h_y + 1$. The second sublist stores all "non-pushable" edges $\{x, y\}$ incident to x. These are edges $\{x, y\}$ such that $(x, y) \notin G^f$ or $h_x \leq h_y$. Given this representation, we have the following three lemmas.

Lemma 8. *It takes constant time to test whether the current flow is a feasible flow. If the flow is feasible, the flow can be extracted from G' in $O(m)$ time. If the flow is not feasible yet, then it takes constant time to pick a vertex x with positive excess, to decide whether to relabel x or push flow along one of its out-edges in G^f, and to select an edge (x, y) along which to push flow.*

Proof. By Lems. 1, 2 and 4, the edge labelling f is a preflow at all times. Thus, it is a flow once there are no vertices left with positive excess. This is easily tested in constant time by checking whether the list of vertices with positive excess is empty. If it is not, then we can choose x to be the first vertex in this list, which takes constant time. Given x, we inspect its "pushable" adjacency list. If it is empty, which can be tested in constant time, then x can be relabelled. Otherwise, we select the first edge $\{x, y\}$ in this list, again in constant time. The edge (x, y) satisfies the conditions to push flow along it in this case.

Once the flow is feasible, extracting it requires iterating over the edges of G'. Each edge $\{x, y\}$ stores which of the edges (x, y) and (y, x) belong to G, as well as the flow along them. Thus, the flow along these two edges can be copied to an output list of flow values along all edges of G in constant time. Since G' has at most m edges, this takes $O(m)$ time for all edges of G'. □

Lemma 9. *A* Push *operation takes constant time.*

Proof. Given a vertex x with positive excess and a pushable edge $\{x, y\}$ incident to it, x stores its excess e_x, and $\{x, y\}$ stores the necessary information to compute $u^f_{x,y} = u_{x,y} - f_{x,y} + f_{y,x}$. The edge $\{x, y\}$ also stores a pointer to y, as part of the adjacency list representation, so y can be found in constant time. We can thus compute $\delta = \min\left(e_x, u^f_{x,y}\right)$ and update the flow and excess values stored with x, y, and $\{x, y\}$ as follows to reflect moving δ units of flow from x to y:

$$\begin{aligned} e_x &= e_x - \delta \\ e_y &= e_y + \delta && \text{(unless } y \in \{s, t\}\text{)} \\ f_{y,x} &= f_{y,x} - \min(\delta, f_{y,x}) \\ f_{x,y} &= f_{x,y} + \max(0, \delta - f_{y,x}) \end{aligned}$$

Then we test whether e_x is now 0. If so, we remove x from the list of vertices with positive excess and

add it to the list of vertices with excess 0. Since these lists are represented as doubly-linked lists, this takes constant time. Similarly, if e_y was 0 before the PUSH operation and has positive excess now, then y needs to be moved from the list of vertics with excess 0 to the list of vertices with positive excess, which can be accomplished in constant time.

If $u^f_{x,y} = 0$ now, then $\{x, y\}$ needs to be moved from x's list of "pushable" edges to the list of "non-pushable" edges. Again, since these two lists associated with x are represented as doubly linked lists, this can be accomplished in constant time. Sinc $h_x = h_y + 1$, the list $\{x, y\}$ remains non-pushable from y's perspective, so no change is needed at y's end.

Overall, all of these steps take constant time. □

LEMMA 10. *A* RELABEL *operation takes* $O(\deg(x))$*, time, where* x *is the vertex being relabelled.*

Proof. Given a vertex x to be relabelled, it takes $O(\deg(x))$ time to inspect all its incident edges in G'. For each such edge $\{x, y\}$, we can calculate $u^f_{x,y} = u_{x,y} - f_{x,y} + f_{y,x}$ in constant time. If $u^f_{x,y} > 0$, then $(x, y) \in G^f$; otherwise, it isn't. Thus, we can determine the out-edges of x in G^f in $O(\deg(x))$ time. For each such edge (x, y), we inspect y, in constant time, to retrieve h_y. Thus, we can calculate the new value of h_x,

$$h_x = 1 + \min\left\{h_y \,\middle|\, (x, y) \in G^f\right\},$$

in $O(\deg(x))$ time.

After updating h_x, some out-edges of x in G^f may become "pushable" and some in-edges of x in G^f may cease to be "pushable". We inspect each of the edges incident to x in G'. For each edge $\{x, y\}$, we test in constant time whether $h_x = h_y + 1$, whether $h_y = h_x + 1$, whether $u^f_{x,y} > 0$, and whether $u^f_{y,x} > 0$. This allows us to test which of the edges (x, y) and (y, x) is a "pushable" edge of G^f, and we move $\{x, y\}$ to the "pushable" or "non-pushable" portions of the adjacency lists of x and y accordingly. This takes constant time per edge $\{x, y\}$, $O(\deg(x))$ time in total. □

4.2 THE TOTAL COST OF RELABEL OPERATIONS

We never relabel s or t (because they never have positive excess). Every vertex $x \in V \setminus \{s, t\}$ starts at height $h_x = 0$. By Lem. 3, every time x is relabelled, h_x increases. The following lemma will help us prove a bound on h_x and, thereby, a bound on how often x is relabelled over the course of the algorithm.

LEMMA 11. *If* x *has positive excess, then there exists a path from* x *to* s *in* G^f*.*

Proof. Consider the set Y of all vertices y such that G^f does not contain any path from y to s. Since t never has any excess, we never push any flow along any of its out-edges. Thus, the net in-flow of t is

$$\sum_{x \in V} f_{x,t} \geq 0.$$

For any vertex $y \in Y \setminus \{t\}$, its net in-flow is

$$\sum_{x \in V} (f_{x,y} - f_{y,x}) = e_y \geq 0$$

because $s \notin Y$ and, by Lems. 1, 2 and 4, f is always a preflow. Thus, no matter whether $t \in Y$ or not, we

have

$$\sum_{y\in Y}\sum_{x\in V}(f_{x,y}-f_{y,x}) \geq \sum_{y\in Y\setminus\{t\}} e_y > 0$$

if there exists a vertex y with positive excess that does not have a path to s in G^f. However, we have

$$\sum_{y\in Y}\sum_{x\in V}(f_{x,y}-f_{y,x}) = \sum_{y\in Y}\sum_{x\in Y}(f_{x,y}-f_{y,x}) + \sum_{y\in Y}\sum_{x\in V\setminus Y}(f_{x,y}-f_{y,x}).$$

The first sum on the right-hand side is 0 because every edge between vertices in y is counted twice, once with positive sign and once with negative sign. Thus, since $f_{v,w} \geq 0$, for every edge (v,w), we have

$$\sum_{x\in V\setminus Y}\sum_{y\in Y} f_{x,y} \geq \sum_{x\in V\setminus Y}\sum_{y\in Y}(f_{x,y}-f_{y,x}) > 0.$$

Therefore, since $f_{v,w} \geq 0$, for every edge (v,w), there exists an edge (x,y) with $x \in V\setminus Y$, $y \in Y$, and $f_{x,y} > 0$. Since $x \in V\setminus Y$, there exists a path P from x to s in G^f. Since $f_{x,y} > 0$, G^f contains the edge (y,x). Thus, $\langle y\rangle \circ P$ is a path from y to s in G^f, a contradiction because $y \in Y$. This shows that there cannot exist a vertex $y \in Y$ with positive excess, that is, all vertices with positive excess have a path to s in G^f. □

LEMMA 12. *For every vertex $x \in V\setminus\{s,t\}$, $h_x < 2n$ at all times.*

Proof. Initially, every vertex $x \neq s$ has height 0. The height of a vertex $x \in V\setminus\{s,t\}$ changes only when x is relabelled. At this time, x has positive excess. Thus, by Lem. 11, there exists a path $P = \langle x = x_0, \ldots, x_k = s\rangle$ from x to s in G^f at this time. By Lems. 1, 2 and 4, h is a valid height function at all times. Thus, we have $h_{x_i} \leq h_{x_{i+1}} + 1$, for all $0 \leq i < k$, and, therefore $h_x \leq h_s + k$ after every RELABEL operation. Since P contains at most n vertices, we have $k \leq n-1$, so $h_x \leq h_s + n - 1 = 2n-1$. □

Lem. 12 is all we need to bound the cost of RELABEL operations.

LEMMA 13. *The total cost of all* RELABEL *operations is $O(nm)$.*

Proof. By Lem. 3, every time we relabel a vertex x, its height increases. Initially, x has height $h_x = 0$. By Lem. 12, h_x never exceeds $2n-1$. Thus, x is relabelled at most $2n-1$ times. By Lem. 10, relabelling x takes $O(\deg(x))$ time. Thus, the cost of all RELABEL operations applied to x is $O(n\deg(x))$. By summing over all vertices in G, we obtain that the total cost of all RELABEL operations is

$$\sum_{x\in V} O(n\deg(x)) = O\left(n\cdot\sum_{x\in V}\deg(x)\right) = O(nm)$$

because $\sum_{x\in V}\deg(x) = 2m$. □

4.3 THE TOTAL COST OF PUSH OPERATIONS

To bound the total cost of PUSH operations, we split them into two types. We call a PUSH operation **saturating** if it saturates the edge (x,y) along which it pushes flow, that is, if $u^f_{x,y} = 0$ after the PUSH operation. This implies that $(x,y) \notin G^f$ after the PUSH operation. If $u^f_{x,y} > 0$ after the PUSH operation,

then the PUSH operation is **non-saturating**. We determine the costs of these two types of PUSH operations separately.

4.3.1 SATURATING PUSH OPERATIONS

LEMMA 14. *For every edge $(x, y) \in G^f$, the push-relabel algorithm performs at most n saturating* PUSH *operations along this edge.*

Proof. Assume we perform k saturating PUSH operations along the edge (x, y), and let $h_x^{(i)}$ and $h_y^{(i)}$ be the values of h_x and h_y, respectively, before the ith such PUSH operation. We prove that $h_x^{(i+1)} \geq h_x^{(i)} + 2$, for all $1 \leq i < k$. Then, since $h_x^{(1)} \geq 0$, this shows that $h_x^{(k)} \geq 2k - 2$. However, by Lem. 12, we have $h^{(k)} < 2n$. Therefore, $k \leq n$.

Consider the ith saturating PUSH operation along the edge (x, y). Since this operation is saturating, we have $(x, y) \notin G^f$ immediately after this operation. Immediately before the $(i+1)$st saturating PUSH operation along the edge (x, y), we have $(x, y) \in G^f$ again. Thus, the edge (x, y) must be reintroduced to G^f by pushing flow along the edge $(y, x) \in G^f$ between the ith saturating PUSH operation and the $(i+1)$st saturationg PUSH operation along the edge (x, y). Let h'_x and h'_y be the values of h_x and h_y, respectively, before this PUSH operation along the edge (y, x).

Since we can only push along edges (v, w) with $h_v = h_w + 1$, we have

$$\begin{aligned} h'_y &= h'_x + 1 \text{ and} \\ h_x^{(i+1)} &= h_y^{(i+1)} + 1. \end{aligned}$$

By Lem. 3, we have

$$\begin{aligned} h'_x &\geq h_x^{(i)} \text{ and} \\ h_y^{(i+1)} &\geq h'_y. \end{aligned}$$

Thus,

$$\begin{aligned} h_x^{(i+1)} &= h_y^{(i+1)} + 1 \\ &\geq h'_y + 1 \\ &= h'_x + 2 \\ &\geq h_x^{(i)} + 2. \end{aligned}$$

□

Since G has m edges and, for every edge (x, y) of G, G^f can contain the edge (x, y) or (y, x) or both, Lem. 14 implies that the push-relabel algorithm performs at most $2nm$ saturating PUSH operations in total. By Lem. 9, the cost per PUSH operation is constant. Thus, we immediately obtain that

COROLLARY 15. *The total cost of saturating* PUSH *operations is $O(nm)$.*

4.3.2 NON-SATURATIONG PUSH OPERATIONS

Counting non-saturating PUSH operations is the tricky part. We prove that

LEMMA 16. *The push-relabel algorithm performs at most $2n^2(2m + 1)$ non-saturating* PUSH *operations.*

Proof. To prove this, we define a potential function

$$\Phi = \sum_{e_x > 0} h_x.$$

In words, Φ is the total height of all vertices with positive excess. Initially, $\Phi = 0$ because every vertex other than s has height zero and $e_s = 0$. Moreover, since every vertex has a non-negative height, Φ is never negative.

The proof idea now is to bound the increase in potential resulting from every RELABEL and saturating PUSH operation and to show that every non-saturating PUSH operation decreases Φ by at least 1. Specifically, we show that RELABEL and saturating PUSH operations add less than $2n^2(2m+1)$ in total to Φ. Since every non-saturating PUSH operation decreases Φ by at least one and Φ is never negative, this shows that there are at most $2n^2(2m+1)$ non-saturating PUSH operations.

RELABEL operations. Consider all RELABEL operations applied to some vertex x. Assume there are k such operations, let $h_x^{(0)} = 0$ be the height of x before the first such operation, and let $h_x^{(i)}$ be the height of x immediately after the ith such operation. Since relabelling x does not change the excess of any vertex and changes only the height of x, the increase in potential from the ith RELABEL operation applied to x is $h_x^{(i)} - h_x^{(i-1)}$. Thus, the total increase in potential from relabelling x is

$$\sum_{i=1}^{k} \left(h_x^{(i)} - h_x^{(i-1)}\right) = h_x^{(k)} - h_x^{(0)} = h_x^{(k)} < 2n,$$

by Lem. 12. The total potential increase from relabelling all vertices is thus less than $2n^2$.

Saturating PUSH operations. A saturating PUSH operation along an edge (x, y) does not alter the height of any vertex, may or may not reduce x's excess to 0, and results in y having positive excess after the PUSH operation. Thus, the greatest increase in potential is achieved if $e_x > 0$ before and after the PUSH operation, and $e_y = 0$ before the PUSH operation. In this case, x's contribution to Φ does not change, and y's contribution to Φ increases by h_y. Since $h_y < 2n$, by Lem. 12, this shows that a single saturationg PUSH operation increases Φ by less than $2n$. As argued in the paragraph before Cor. 15, the push-relabel algorithm performs at most $2nm$ saturating PUSH operations. Thus, the total increase in potential due to saturating PUSH operations is less than $4n^2m$.

Non-saturating PUSH operations. A non-saturating PUSH operation along an edge (x, y) does not alter the height of any vertex, decreases e_x from $e_x > 0$ to $e_x = 0$ and may change e_y from $e_y = 0$ to $e_y > 0$. Thus, the decrease in potential is at least $h_x - h_y$. Since we push flow along the edge (x, y), we have $h_x = h_y + 1$ at the time of the PUSH operation. Thus, Φ decreases by at least 1. Since RELABEL and saturating PUSH operations increase the potential by less then $2n^2(2m+1)$ and every non-saturating PUSH operation decreases the potential by at least 1, we obtain a bound of $2n^2(2m+1)$ on the total number of non-saturating PUSH operations. □

Since each PUSH operation takes constant time, we immediately obtain that

COROLLARY 17. *The total cost of non-saturating* PUSH *operations is* $O\left(n^2m\right)$.

4.4 SUMMARY

By Lem. 13 and Cors. 15 and 17, the total cost of all PUSH and RELABEL operations is $O\left(n^2m\right)$. Thus, this is also a bound on the number of these operations we perform. By Lem. 8, choosing the vertex to be relabelled or the edge along which to push flow takes constant time per operation. Together with Prop. 7, this shows that

THEOREM 18. *The push-relabel algorithm computes a maximum flow in $O\left(n^2m\right)$ time.*

Note that non-saturating PUSH operations constitute the bottleneck of the algorithm: without them, the algorithm would take only $O(nm)$ time. The variants of the push-relabel algorithm with running times $O\left(n^3\right)$ and $O\left(n^2\sqrt{m}\right)$ achieve these faster running times by choosing carefully when to relabel and when to push, and which vertex to relabel or along which edge to push. This does not affect the cost of RELABEL or saturating PUSH operations but reduces the number non-saturating PUSH operations to $O\left(n^3\right)$ and $O\left(n^2\sqrt{m}\right)$, respectively.

EXERCISES

EXERCISE 1. Prove that if the capacities of all edges are integers, then the Ford-Fulkerson, Edmonds-Karp, and push-relabel algorithms all compute an integral flow, that is, a flow f such that $f_{x,y} \in \mathbb{N}_0$, for every edge (x, y) in the network. Conclude that this proves that, if the capacities of all edges are integers, then there exists an integral maximum flow.