Chapter 3
Maximum Flow by Preflow-Push

3.1 Pre-flows and Distance Labellings

The Augmenting Path Algorithm of Chapter 2 maintains a feasible flow at each stage. The value of the flow is steadily increased until a saturated cut has been formed. This is the signal that the flow has a maximum value. An alternate approach is used by the preflow-push algorithms. These algorithms maintain at all stages a feasible pre-flow (a "flow" without flow conservation) that has a saturated cut. The pre-flow is changed step by step until it does satisfy flow conservation. The resulting flow then has a saturated cut so is a maximum flow.

Suppose that N is a network with edge capacities c_{ab} and $\mathcal{P} = \{p_{ab}\}$ is an assignment of flow to the edges of N. Then $\mathcal{P}$ is a *preflow* if at each internal node $v \neq s, t$, we have $\sum_{h(e)=v} p_e \geq \sum_{t(e)=v} p_e$. Thus we assume that, at each internal node, the flow in is at least as large as the flow out. The *excess flow* at v is defined by

$$ex(v) = \sum_{h(e)=v} p_e - \sum_{t(e)=v} p_e = \sum_w p_{wv} - \sum_u p_{vu}.$$

Of course, if $ex(v) = 0$ for all internal nodes v then $\mathcal{P}$ is a flow. We call $\mathcal{P}$ a *feasible preflow* if, in addition, $0 \leq p_e \leq c_e$. Our general strategy is to start with an initial preflow then successively modify the flows on various edges so that at all times the preflow is feasible and moves steadily toward satisfying flow conservation.

Suppose we have a network N with a preflow $\mathcal{P} = \{p_{ab}\}$. If there is an internal node with positive excess $ex(a) > 0$ then we would like to push this flow away from a and toward the sink t. If we can somehow push all the excess flows to the sink then we will have a maximum flow. If we want to push excess flow from a to b then we can do it forward along ab if $p_{ab} < c_{ab}$ or backwards along ba if $p_{ba} > 0$ or both. A *push on* ab is performed by moving up to $ex(a)$ units of flow from a to b with up to $c_{ab} - p_{ab}$ units moving forward on ab and up to p_{ba} units moving backward on ba. Once a push along ab has been performed, the excess $ex(a)$ at a will decrease while $ex(b)$ will increase.

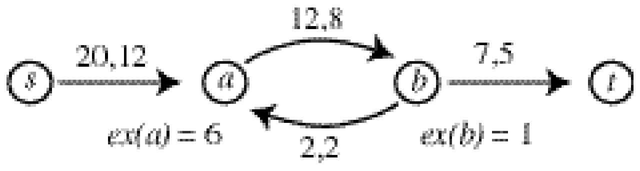


Figure 3.1 A simple preflow

As an example, consider the simple network in Figure 3.1. There are 6 excess units at a. We can push all 6 units along ab with 4 units moving foward along ab and 2 units moving backward along ba. The result is shown in Figure 3.2. The preflow now satisfies flow conservation at a but not at b.

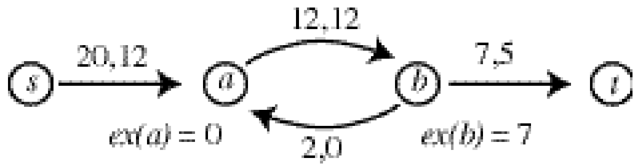


Figure 3.2 The preflow after pushing on ab

There might be several different ways to perform a push along ab. If, in the example, we were only pushing away an excess of 3 units there would not be enough new flow both to saturate ab and to empty ba; there would be a choice to make. Moreover, the push on ab might not exhaust the excess at a so that further pushes are needed at a. The simple network fragment in Figure 3.3 illustrates the point. To move all 7 excess units away from a will require pushes on several edges. An actual implementation will have to include rules that decide which edges to use and in what order and also which nodes to work on.

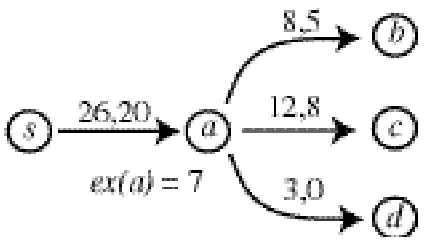


Figure 3.3 A preflow in a network fragment

Suppose we start with a preflow in our network and randomly push excess flow away from nodes. So far there is nothing to stop us from pushing the excess flow around in a cycle or even straight back where it came from. Pushing flow is an entirely local event; we need a global tool to direct our progress from preflow to flow.

A *distance labelling* λ assigns to each node v an integer $\lambda(v)$. The conditions that makes $\lambda(v)$ a distance labelling are that: $\lambda(s) = n = |V|$, $\lambda(t) = 0$ and if edge ab is available for a push then $\lambda(a) \leq \lambda(b) + 1$. So we insist that, whenever $p_{ab} < c_{ab}$ or $p_{ba} > 0$ we have $\lambda(a) \leq \lambda(b) + 1$.

For any network N it is possible to define a feasible preflow $\mathcal{P}$ and corresponding distance labelling λ as follows. For every edge sa leading out of the source, set $p_{sa} = c_{sa}$. For all other edges set $p_{ab} = 0$. Certainly $0 \leq p_e \leq c_e$ for all edges e. At an internal node v that is not the head of an edge from the source we have all preflows in and out equal to zero, so $ex(v) = 0$. If, on the other hand, there is an edge sv, then $ex(v) = c_{sv} \geq 0$. Therefore $\mathcal{P}$ is indeed a feasible preflow. Using this preflow, a valid labelling is obtained by setting $\lambda(s) = |V|$ and $\lambda(v) = 0$ for all other vertices. With this definition, $\lambda(a) = \lambda(b)$ unless a or b is the source s. The edges sv are not available for a push since $p_{sv} = c_{sv}$ and $p_{vs} = 0$. Therefore the definition of distance labelling is satisfied by λ.

Figure 3.4 illustrates these ideas. The edge labels are in the order c_e, p_e and the numbers at the vertices form the distance labelling.

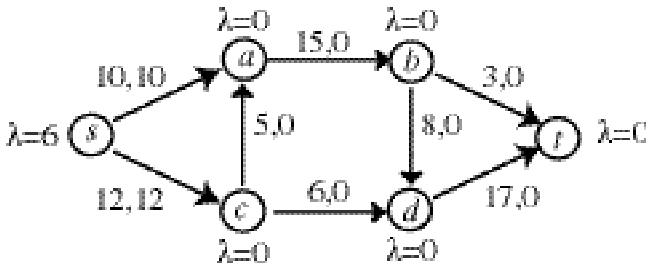


Figure 3.4 The Initial Preflow and Labelling of a Network

In Figure 3.4, the partition $S = \{s\}, T = \{a, b, c, d, t\}$ defines a saturated cut since all the edges coming out of the source are saturated and had there been an edge directed into the source it would be empty. As mentioned earlier, the existence of such a saturated cut is a general phenomenon. We complete this section with a proof of this fact.

THEOREM 3.1 *Suppose that network N has a feasible preflow $\mathcal{P}$ and a distance labelling λ. Then there is a cut (S, T) with every edge of $\delta(S)$ saturated and every edge of $\delta(T)$ empty*

Proof. Since there are $n = |V|$ nodes in the network then amongst the $n + 1$ integers $0 \ldots n$, there is at least one integer k which is not used in the labelling. Let $S = \{v|\lambda(v) > k\}$ and $T = V \setminus S$. So $s \in S$ and $t \in T$. Also, for any edge $e = ab$ with $a \in S$ and $b \in T$, we have $\lambda(a) > k > \lambda(b)$. But then $\lambda(a) > \lambda(b) + 1$. By the definition of a distance labelling, this edge must be not admissible. Hence ab is saturated and ba is empty as required. □

This theorem says that any feasible preflow with a distance labelling has a saturated cut. For the special case of a flow, the Max-Flow Min-Cut Theorem then gives the following corollary which establishes the stopping condition for the preflow-push algorithms.

THEOREM 3.2 *If a feasible flow F has a distance labelling then F is a maximum flow.*

3.2 Preflow Push Algorithms

In this section, the ideas of preflow and distance labelling on a network will be fashioned into an algorithm for the maximum flow problem called the Preflow Push Algorithm. In fact, several of the details required for implementing Preflow Push can be specified in different ways so this can be considered a class of algorithms based on a central theme that we develop here.

We have a network N with a preflow $\mathcal{P}$ and a distance labelling λ. At each internal node v there is some excess flow $ex(v)$. As before we denote the capacity of the arc ab by c_{ab} while the preflow on this edge is p_{ab}.

We will call an edge ab an *available edge* if either ab or ba is an arc of N and either $p_{ab} < c_{ab}$ or $p_{ba} > 0$ or both. Thus an available edge is a candidate for a push operation. Note that the availability of an edge depends on the preflow; as the preflow is changed a given edge ab may gain or loose this status. The maximum amount that can be pushed along an available edge ab is $\tilde{c}_{ab} = c_{ab} - p_{ab} + p_{ba}$. The key to controlling the pushing of flow is to define an available edge ab to be *admissable* if and only if $\lambda(a) = \lambda(b)+1$. The Preflow Push Algorithm allows pushes only on admissable edges.In Figure 3.5, edges va, vb, vc and vd are available but only vb and vc are admissable. Note also that if we shift attention to a the edge av is admissible. From the definition, if an edge uv is admissible then edge vu is definitely not. Thus the labelling tells us a direction to push flow.

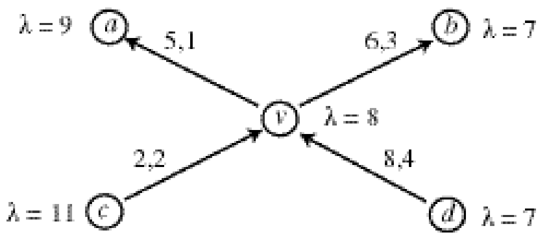


Figure 3.5 Available and admissible edges at v

The final ingredient for the algorthmn is *relabelling*. If, in Figure 3.5, we want to push excess flow from v to a, this will not be possible unless the labels change so that va is admissable, that is $\lambda(v) = \lambda(a) + 1$. Therefore some relabelling procedure is required.

The Preflow Push algorithm is based on the following local routine that operates on the excess flow at a single vertex.

Process(v)

1. Start with an active node v (so $ex(v) > 0$).
2. Pick an admissible edge va.
3. Push $min\{ex(v), \tilde{c}_{va}\}$ additional flow along va.
4. Repeat Steps 2 and 3 until either $ex(v) = 0$ or there are no more admissible edges.

5. If $ex(v) > 0$ and there are no admissable then relabel v as follows. Set

$$\lambda(v) = min\{\lambda(a) + 1 | va \text{ is an available edge}\}$$

This will produce at least one admissible edge; return to Step 2. Note that since $ex(v) > 0$ this excess flow must have arrived on some arcs that are now available for a push.

6. If $ex(v) = 0$ then v is no longer active and we are finished processing v.

With this routine in hand, the Preflow Push algorithm can be simply stated.

Preflow Push Algorithm

1. Initialize: define an initial preflow and distance labelling on N
2. While an active node v remains: Process(v).
3. The preflow is now a maximum flow.

Example 3.1

The example in Figure 3.6 is an extremely simple, indeed linear, network. It is transparent that the maximum flow value is 3. Nevertheless, it is a useful first example of pushing and relabelling.

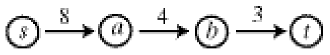


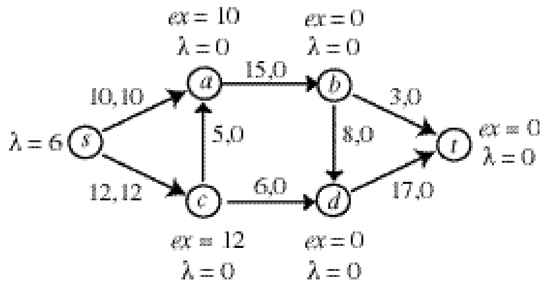
Figure 3.6 A simple example (Edge labels are capacities.)

		$\lambda(s)$	p_{sa}	$ex(a)$	$\lambda(a)$	p_{ab}	$ex(b)$	$\lambda(b)$	p_{bt}	$ex(t)$	$\lambda(t)$
initialization		4	8	8	0	0	0	0	0	0	0
process(a)	relabel a	4	8	8	1	0	0	0	0	0	0
	push on ab	4	8	4	1	4	4	0	0	0	0
	relabel a	4	8	4	5	4	4	0	0	0	0
	push on as	4	4	0	5	4	4	0	0	0	0
process(b)	relabel b	4	4	0	5	4	4	1	0	0	0
	push on bt	4	4	0	5	4	1	1	3	3	0
	relabel b	4	4	0	5	4	1	6	3	3	0
	push on ba	4	4	1	5	3	0	6	3	3	0
process(a)	push on as	4	3	0	5	3	0	6	3	3	0
STOP: maximum flow			3			3			3		

Example 3.2

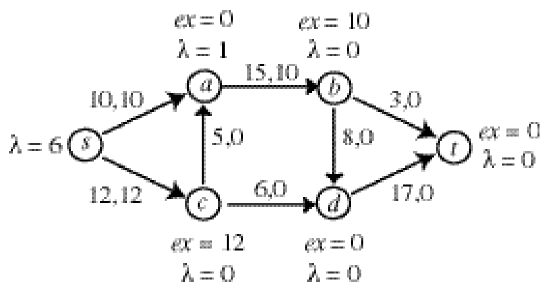
As a second example, we will use Preflow Push to determine a maximum flow in the network of Figure 3.4. In this example, the next node to be processed is choosen arbitrarily among the current active nodes.

1. Initialize the preflow and labelling.



2. Relabel a: $\lambda(a) = 1$

3. Push on ab: $p_{ab} = 10, ex(a) = 0$



4. Relabel c: $\lambda(c) = 1$

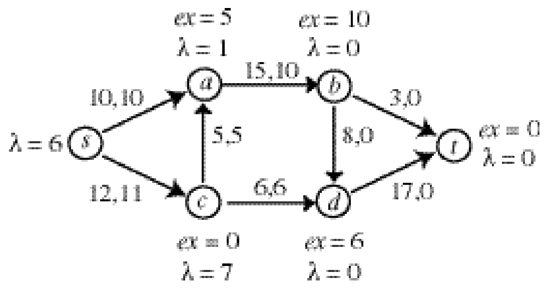
5. Push on cd: $p_{cd} = 6, ex(c) = 6$

6. Relabel c: $\lambda(c) = 2$

7. Push on ca: $p_{ca} = 5, ex(c) = 1$

8. Relabel c: $\lambda(c) = 7$

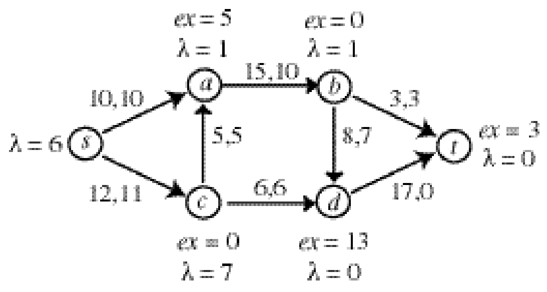
9. Push on cs: $p_{sc} = 11, ex(c) = 0$



10. Relabel b: $\lambda(b) = 1$

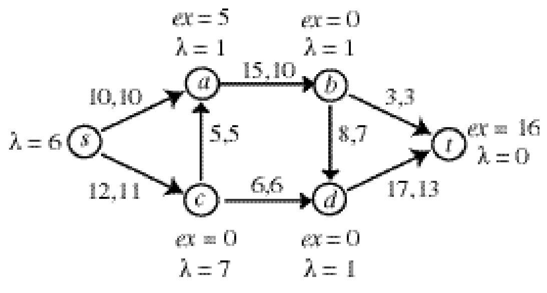
11. Push on bt: $p_{bt} = 3, ex(b) = 7$

12.Push on bd: $p_{bd} = 7, ex(b) = 0$



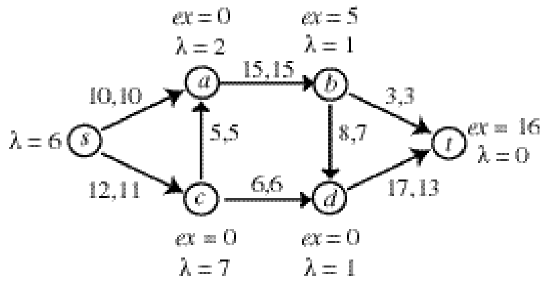
13. Relabel d: $\lambda(d) = 1$

14. Push on dt: $p_{dt} = 13, ex(d) = 0$

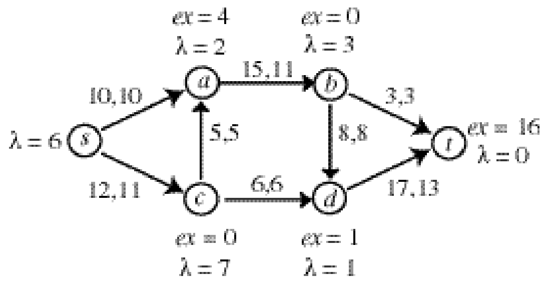


15. Relabel a: $\lambda(a) = 2$

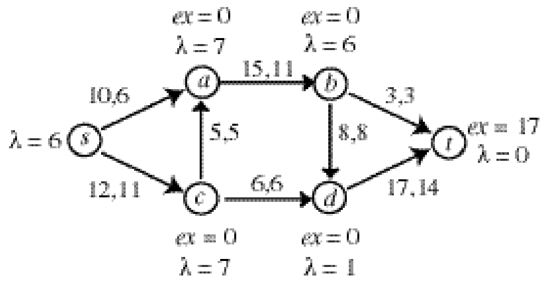
16. Push on ab: $p_{ab} = 15, ex(a) = 0$



17. Relabel b: $\lambda(b) = 2$
18. Push on bd: $p_{bd} = 8, ex(b) = 4$
19. Relabel b: $\lambda(b) = 3$
20. Push on ba: $p_{ab} = 11, ex(b) = 0$



21. Push on dt: $p_{dt} = 14, ex(d) = 0$
22. Relabel a: $\lambda(a) = 4$
23. Push on ab: $p_{ab} = 15, ex(a) = 0$
24. Relabel b: $\lambda(b) = 5$
25. Push on ba: $p_{ab} = 11, ex(b) = 0$
26. Relabel a: $\lambda(a) = 6$
27. Push on ab: $p_{ab} = 15, ex(a) = 0$
28. Relabel b: $\lambda(b) = 6$
29. Push on ba: $p_{ab} = 11, ex(b) = 0$
30. Relabel a: $\lambda(a) = 7$
31. Push on as: $p_{sa} = 6, ex(a) = 0$
32. STOP: No active vertices remain.



On such a small example it is easy to see ways the algorithm could have taken a short cut. For example, the relabelling loop between nodes a and b is a candidate. The problem is that the distance labels are our global control, but we are modifying them locally. One way of improving performance is to periodically calculate a new distance labelling for the entire network. This is discussed further in the next section.

Before leaving this example, locate in each of the displayed networks the saturated cut guaranteed by Theorem 3.1.

3.3 Performance

Correctness and Worst-case Analysis

In order to show that the Preflow-Push algorithm is correct, we must show that the algorithm stops in a finite number of steps with a maximum flow. Since we can estimate the number of steps used, the correctness proof also yields a worst-case estimate at the same time. To simplify the discussion, we will analyze the instance of Preflow-Push that always picks an active vertex with maximal label for processing.

There are several facts to be checked. First, we must show that the algorithm maintains both a valid feasible preflow and a valid distance labelling at each stage. The next component of the analysis is the proof that, if there are n nodes, then the algorithm takes at most $2n^2$ relabel steps and at most $2n^3$ push steps. Finally, we show that the algorithm doesn't stop while there is an active node. Theorem 3.2 ensures that when the algorithm does stop it has determined a maximum flow.

The argument that the algorithm always maintains a valid feasible preflow is straightforward (Exercise 3.1). The key property of a distance labelling is that $\lambda(a) \leq \lambda(b) + 1$ for any available edge ab. A push step does not change the labels so the only risk to the distance labelling would be from an edge becoming available. Suppose that some push step changes the status of an edge ab from unavailable to available. It must have been a push along ba that did this so edge ba was admissable before that push. Hence $\lambda(b) = \lambda(a) + 1$. Then $\lambda(a) = \lambda(b) - 1 < \lambda(b) + 1$ as required. Relabelling is specified by an assignment

$$\lambda(v) = min\{\lambda(a) + 1 | va \text{ is an available edge}\}$$

This ensures that after the relabelling $\lambda(v) \leq \lambda(a) + 1$ for any available edge va. The only other case to checkis an available edge bv. But before the relabel $\lambda(b) \leq \lambda(v) + 1$ which remains valid since the label on v has increased (Exercise 3.2).

The following lemma bounds the number of relabel steps.

Lemma 3.3 *If v is an active node then $\lambda(v) \leq 2n - 1$*

Proof. The key is that if v is an active node then there is a sequence

$$v = v_0 v_1 \cdots v_k = s$$

of nodes leading back to the source with each edge $v_i v_{i+1}$ available. Clearly k can be taken less than n. The result follows from this since the definition of distance labelling says that $\lambda(v_i) \leq \lambda(v_{i+1}) + 1$ for all i so that $\lambda(v) \leq \lambda(s) + k \leq n + (n - 1) = 2n - 1$.

Let S be the set nodes for which such a sequence $v = v_0 v_1 \cdots v_k = s$ exists and let $T = V \setminus S$ be the complement. Our goal is to show that all active nodes are in S or equivalently that there are no active nodes in T. Focus on the sum $\sum_{v \in T} ex(v) \geq 0$. As usual, we expand, change the order of summation and collapse to get

$$0 \leq \sum_{v \in T} ex(v) = \sum_{v \in T} \left(\sum_{a \in V} p_{av} - \sum_{b \in V} p_{vb} \right) = \sum_{a \in S, v \in T} p_{av} - \sum_{v \in T, b \in S} p_{vb} \leq \sum_{a \in S, v \in T} p_{av}.$$

By the definition of S the preflows p_{av} in the last sum are all 0. This forces $\sum_{v \in T} ex(v) = 0$ and therefore all nodes in T are inactive. □

Since there are $n-2$ nodes that can be relabelled and each can be relabelled at most $2n-1$ times the number of relabels steps is at most $(n-1)(2n-1) < 2n^2$.

Having counted the maximum possible number of relabel steps, we next consider the push steps. If you look back at Example 3.2 you will notice that there are two types of pushes used by the algorithm. Some pushes at node a reduce $\tilde{c}_{ab}$ to 0 either by making $p_{ab} = c_{ab}$ or by making $p_{ba} = 0$ while leaving a active. These are called *saturated* pushes. If, on the other hand, $ex(a) < \tilde{c}_{ab}$ then only $ex(a)$ can be pushed, leaving ab available. These are called *unsaturated* pushes. After a saturated push along ab, the edge ab is unavailable; after an unsaturated push along ab, the node a is inactive. It is convenient to count the number of saturated and unsaturated pushes separately.

Lemma 3.4 *If the Preflow-Push algorithm always processes a node with maximal distance label then the number of unsaturated pushes is less than n times the number of relabels*

Proof. Suppose that there is an unsaturated push at a node v. Then this node is now inactive and by assumption $\lambda(v) \geq \lambda(w)$ for any other active node w. Before we can process node v again, more flow will have to be pushed to v. If this flow is pushed from node w, say, then wv is admissible at that moment and node w will have to be relabelled at some point to make $\lambda(w) = \lambda(v) + 1$. This shows that between two unsaturated pushes at node v some node gets relabelled. Since there are n choices for v the result follows. □

Lemma 3.5 *The total number of saturated pushes on a given edge ab is at most n*

Proof. We show that between two saturated pushes on ab the distance label on a increases by at least 2. The result will then follow from Lemma 3.3. Suppose that the algorithm performs a saturated push on ab. Then, by definition, ab is no longer available. Before ab can be available again, there must be a push on ba. We use λ for the distance labels at the time of the first push on ab, and λ' for the labels at the time of the push on ba. Finally we denote by λ'' the distance labels at the time of the next saturated push on ab. Thus $\lambda(a) \leq \lambda'(a) \leq \lambda''(a)$ and $\lambda(b) \leq \lambda'(b) \leq \lambda''(b)$. Then using the definition of an admissible edge we have

$$\lambda''(a) = \lambda''(b) + 1 \geq \lambda'(b) + 1 = \lambda'(a) + 2 \geq \lambda(a) + 2.$$

□

Since there are $n(n-1)$ ordered pairs of nodes potentially defining edges for a push, the total number of saturated pushes is at most $n^2(n-1) < n^3$. Combining all of this analysis gives the following result.

Theorem 3.6 *If the Preflow-Push algorithm is implemented so as to process next a node with maximal label then there are at most $2n^2$ relabel steps and at most $3n^3$ pushes. In particular, the algorithm stops with a maximum flow in a finite number of steps.*

It is only a bit harder to analyze the general Preflow-Push algorithm. See [BILLS] or [GOLDBERG1] for a full acount.

Practical Performance

EXERCISES

3.1 Show that, in the Preflow-Push algorithm each step maintains a valid feasible preflow.

3.2 Show that in the relabel step $\lambda(v) = min\{\lambda(a) + 1 | va$ is an available edge$\}$ the label $\lambda(v)$ increases.

3.3 In Example 3.2, identify both a saturated push and an unsaturated push.