OR 215 **Spring 1999**
Network Flows **M. Hartmann**

THE EXCESS SCALING ALGORITHM FOR THE MAXIMUM FLOW PROBLEM

The Capacity Scaling Algorithm

The Excess Scaling Algorithm

Proof of Polynomial Time

Extensions

THE GENERIC SCALING ALGORITHM

Input: A problem instance ***P***.

- Let ***P**** be a very rough approximation to ***P***.
- Solve problem ***P****, possibly approximately.
- Replace ***P**** by a less rough approximation to ***P***.
- Solve problem ***P****, possibly approximately, starting with the previous solution.

Iterate until problem ***P*** is solved optimally.

IMPROVEMENT IN THE FORD FULKERSON AUGMENTING PATH ALGORITHM

Augment along the path that maximizes $\delta(P)$, the residual capacity of the path.

- Number of augmentations is $O(m \log U)$.
- Running time is $O(m^2 \log U)$.

A scaling variant:

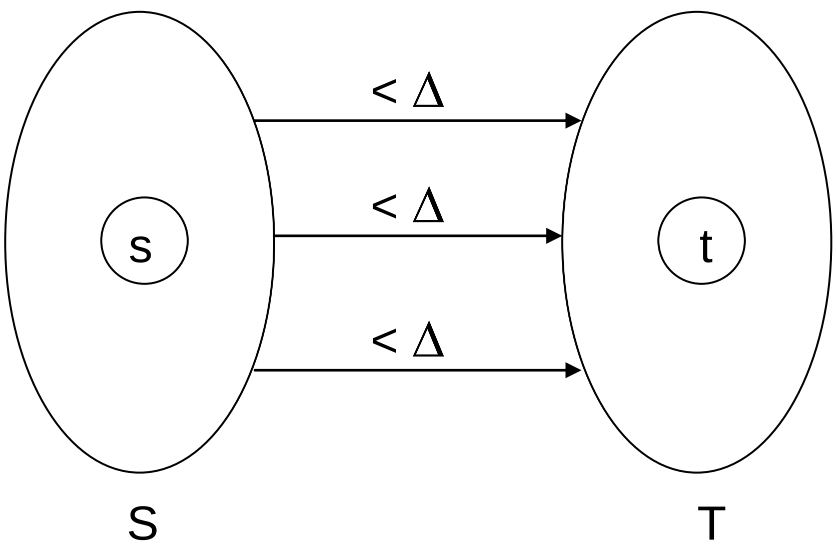
- Select a target value Δ. Initially $\Delta = U/2$.
- Augment along a path p with $\delta(P) \geq \Delta$. If no such path exists, replace Δ by $\Delta/2$.

The number of augmenting paths is O(m log U), or O(m) between successive divisions of Δ by 2.

Running time: O(nm log U) if implemented well, or O(nm) between successive divisions of Δ by 2.

BOUNDING THE NUMBER OF AUGMENTATIONS

At the end of the scaling iteration, the residual capacity from S to T is less than $m\Delta$:



Hence the number of augmentations at next iteration is less than 2m.

AN ALGORITHM FOR WHICH THE NUMBER OF NON-SATURATING PUSHES IS O(n^2 log U).

Let K satisfy $U \le 2^K$ and let $e_{max} = \max\{e(i) : i \in N\}$.

```
algorithm EXCESS-SCALING;
begin
  PREPROCESS;
  K := ⌈log₂ U⌉;
  for k := K down to 0 do
    begin { Δ-scaling phase }
      Δ := 2^k;
      while there is a node i with e(i) > Δ/2 do
      PUSH/RELABEL(i,Δ);
    end;
end
```

PUSH/RELABEL must be modified to ensure that no node excess exceeds Δ in the Δ-scaling phase.

PRELIMINARIES

The Δ in the scaling phase is referred to as the *excess-dominator*. The scaling phase is also called the Δ-scaling phase.

- The number of scaling phases is O(log U).
- At the Δ-scaling phase, $\Delta/2 < e_{max} \leq \Delta$.
- Each scaling phase reduces Δ by a factor of 2.
- After K +1 scaling phases, e_{max} is reduced to 0.

New data structures:

$$\text{ActNode}(\Delta,k) = \{i \in N : d(i) = k, e(i) > \Delta/2\}$$

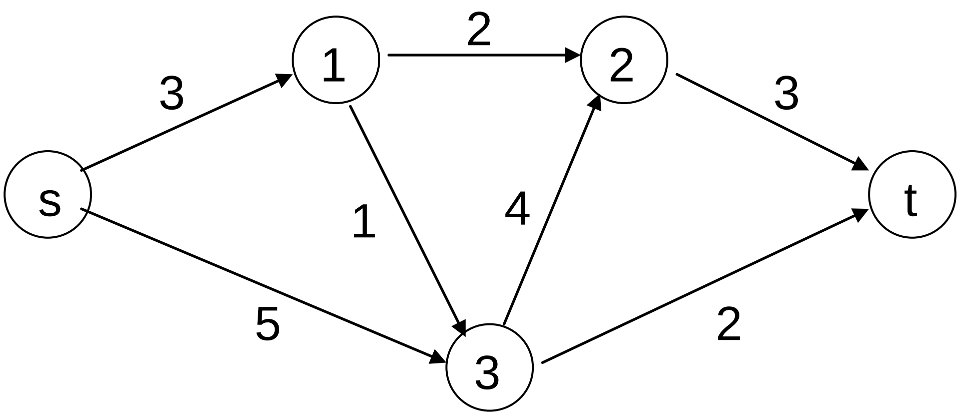
$$\text{ActSet}(\Delta) = \{k : \text{ActNode}(\Delta) \neq \emptyset\}.$$

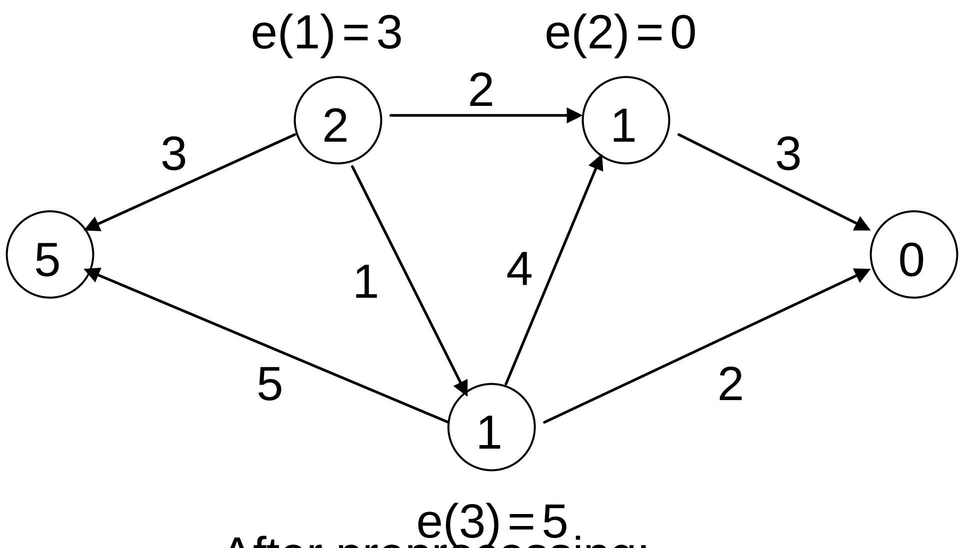
We store the labels in ActSet(Δ) in increasing order.)

```
procedure SELECT;
begin
   if ActSet(Δ) = ∅ then
      go to the Δ/2 scaling phase
   else
      begin
         select the minimum distance label k in ActSet(Δ);
         select  i ∈ ActNode(Δ,k);
         PUSH/RELABEL(i,Δ);
      end;
end
```

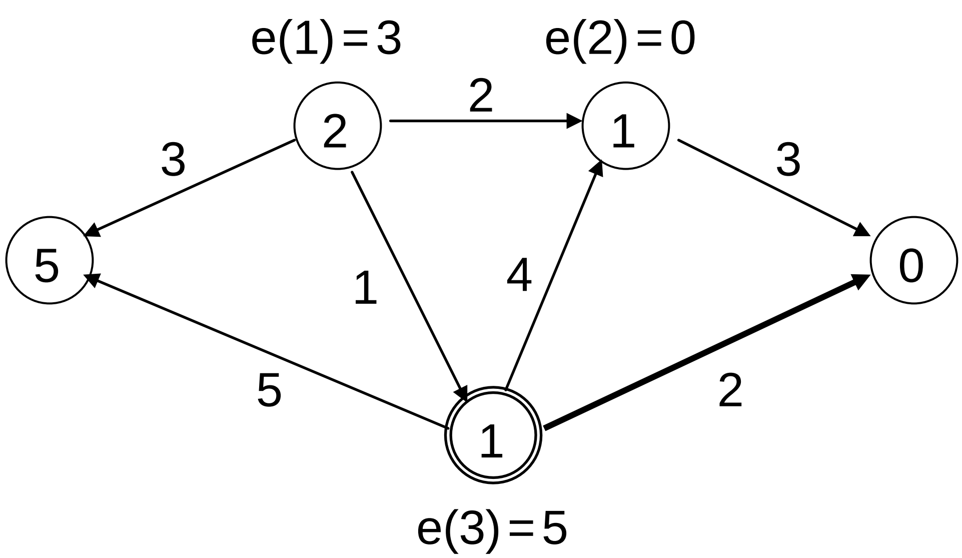
```
procedure PUSH/RELABEL(i,Δ);
begin
   if  there is an admissible arc (i,j)  then
     push  δ := min{e(i),r_ij,Δ-e(j)} units of flow from i  to  j
   else
     d(i) := min{d(j)+1 : (i,j) ∈ A(i) and r_ij > 0};
end
```

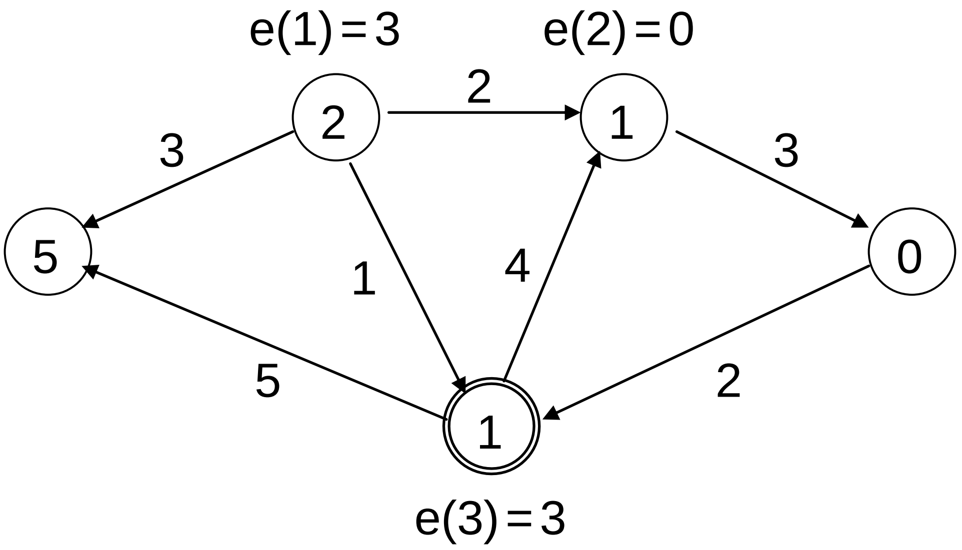
EXAMPLE

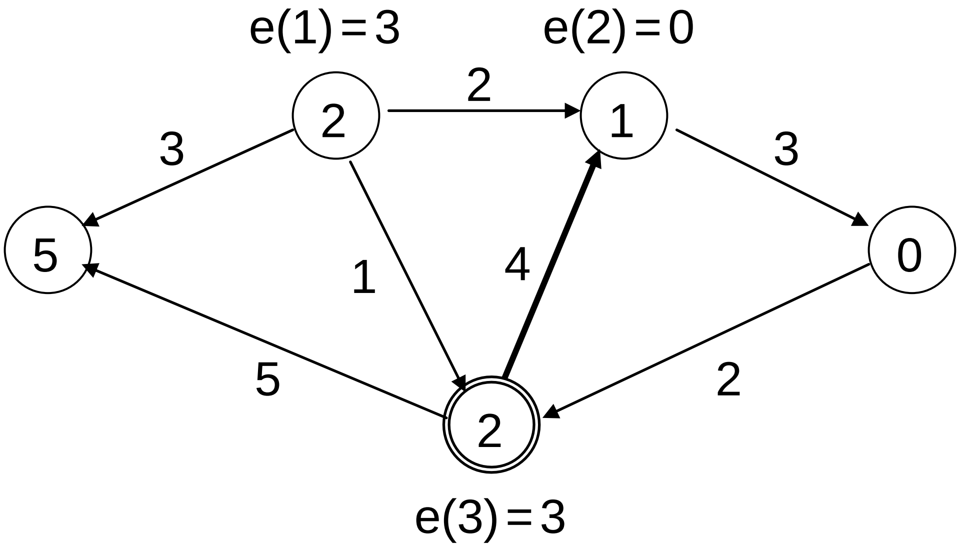


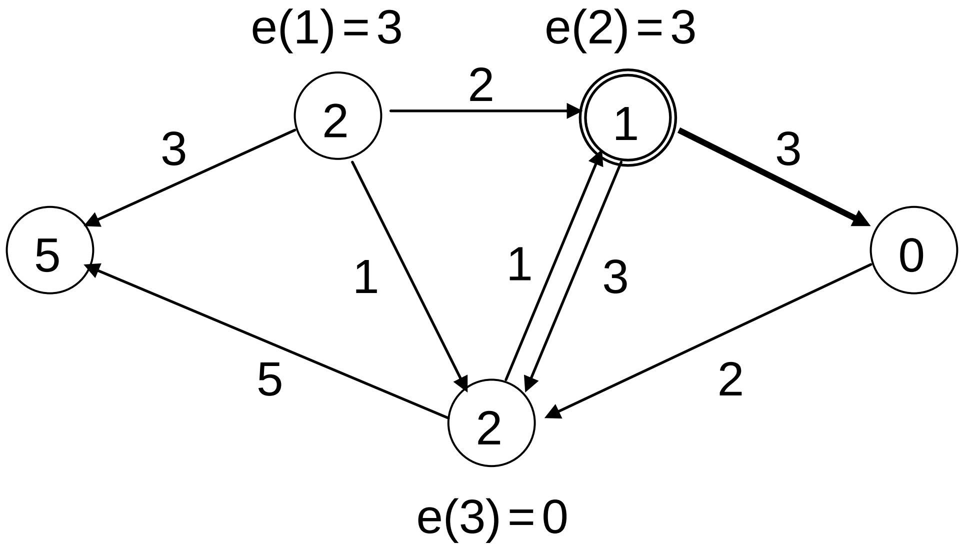


After preprocessing;
nodes are labelled with distances,
arcs are labelled with residual capacities.

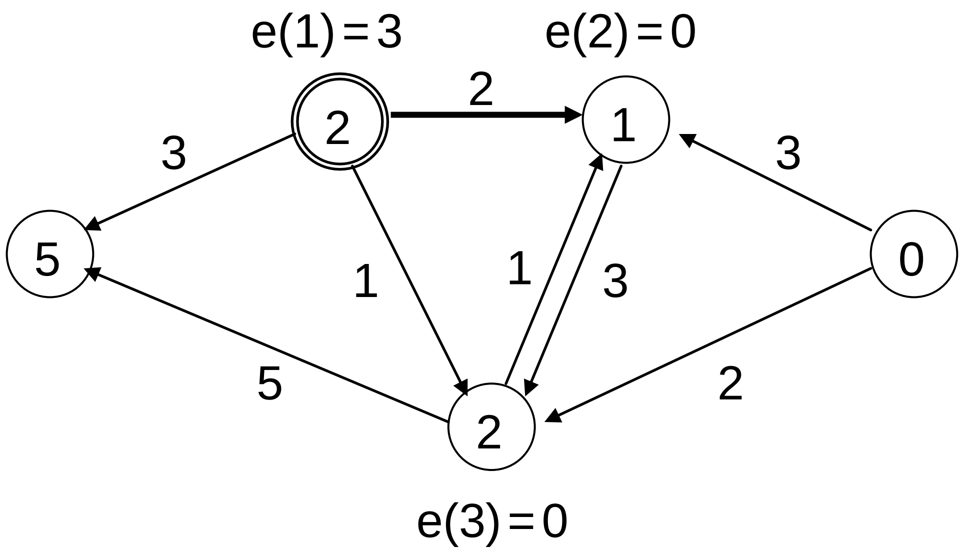



e(1) = 3
e(2) = 0
2
1
3
2
3
5
1
4
0
5
2
1
e(3) = 3

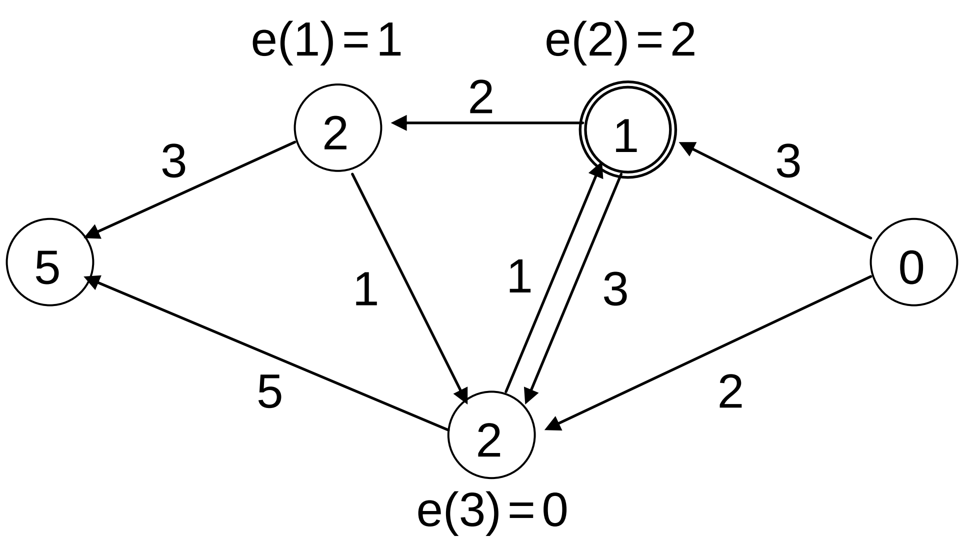

e(1) = 3
e(2) = 0
2
1
3
2
3
5
1
4
0
5
2
2
e(3) = 3



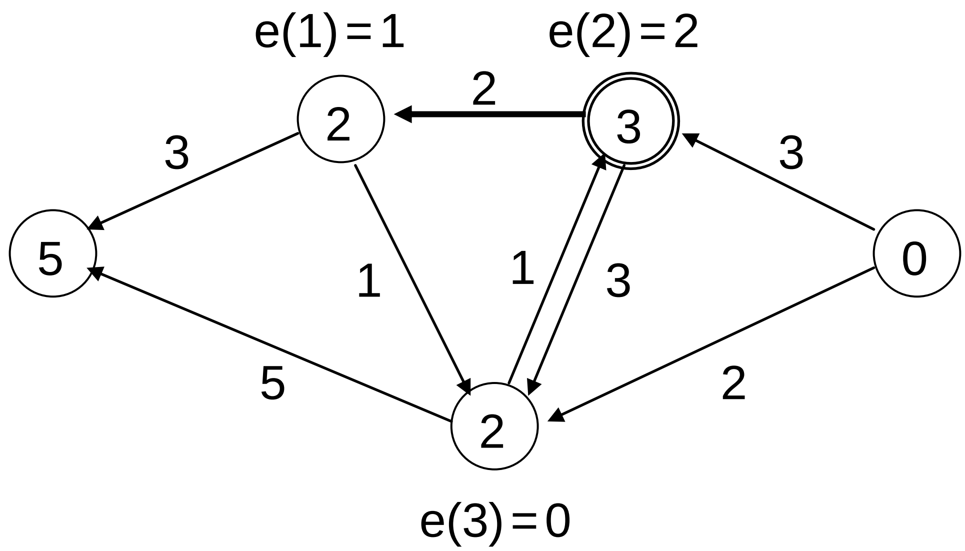
e(1) = 3
e(2) = 3
2
2
1
3
3
5
1
1
3
0
5
2
2
e(3) = 0

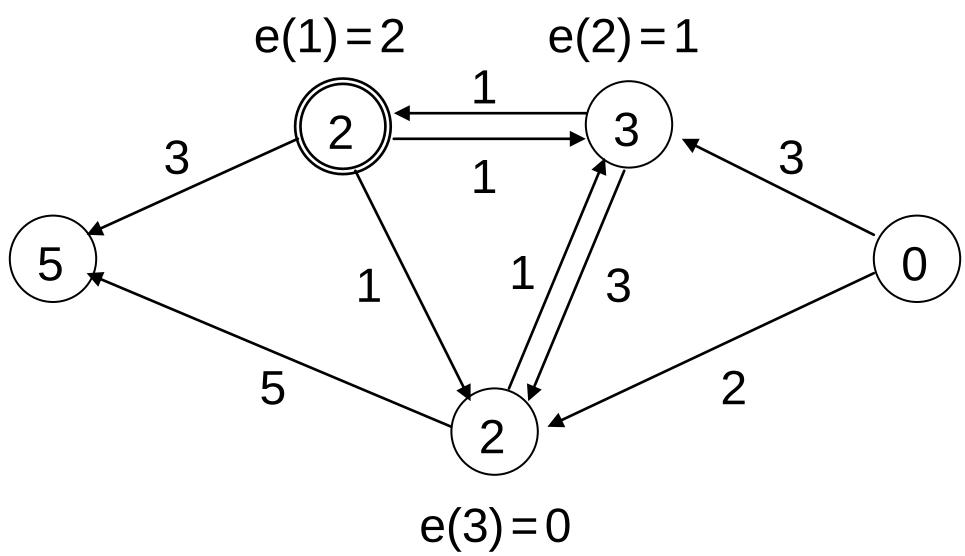


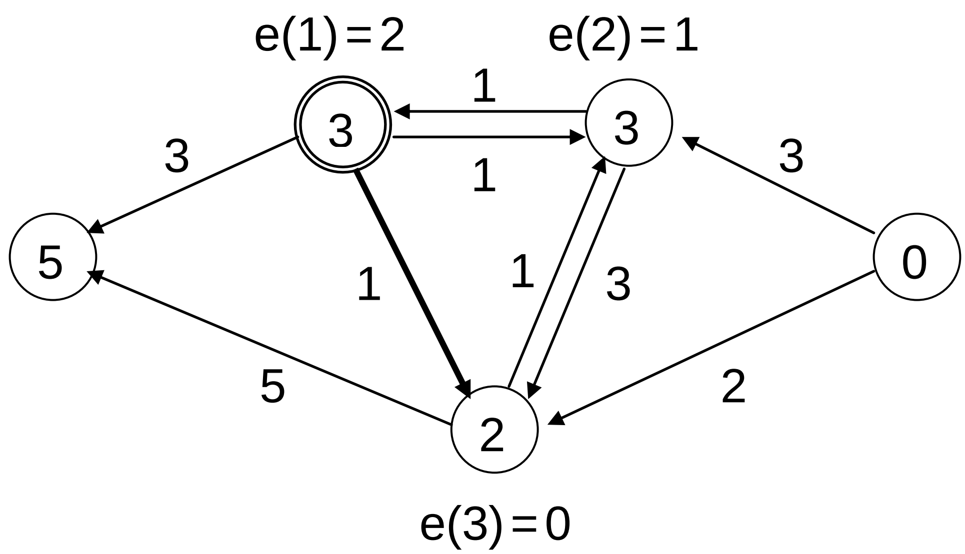
e(1) = 3
e(2) = 0
2
2
1
3
3
5
1
1
3
0
5
2
2
e(3) = 0



e(1) = 1
e(2) = 2
2
2
1
3
3
5
1
1
3
0
5
2
2
e(3) = 0


e(1) = 1
e(2) = 2
e(3) = 0


e(1) = 2
e(2) = 1
e(3) = 0


e(1) = 2
e(2) = 1
e(3) = 0

COMPLEXITY ANALYSIS

Lemma. *The algorithm satisfies the following two conditions:*

1. *Each non-saturating push from a node i to a node j sends at least $\Delta/2$ units of flow.*

2. *No excess ever exceeds Δ.*

Proof: Suppose that we perform a non-saturating push on (i,j). Then $e(i) > \Delta/2$ because $i \in \text{ActNode}(\Delta, d(i))$. Also $e(j) \leq \Delta/2$ because $d(j) \notin \text{ActList}(\Delta)$. [Recall that d(i) has the minimum distance label in ActList(Δ).] Thus $\delta = \min\{e(i), \Delta - e(j)\} \geq \Delta/2$.

Theorem. *The excess-scaling algorithm performs $O(n^2)$ non-saturating pushes per scaling iteration and $O(n^2 \log U)$ pushes in total.*

PROOF. Consider the potential function

$$\Phi = \sum_{i \in N} e(i)d(i)/\Delta.$$

(Think of d(i) as the height of a node, and e(i)/Δ as its weight measured in units of Δ. Then Φ is its gravitational potential.)

Questions:

- What is the effect on Φ of a saturating push? (Does it go up? Does it go down?)

- What is the effect on Φ of a non-saturating push?

- What is the total impact on Φ of the distance increases of a specific node i over all iterations?

A MORE FORMAL PROOF OF THE TIME BOUND

Let ΔD, R, SAT and NS be the steps during which an excess dominator decrease ($\Delta := \Delta/2$), relabel, saturating push or non-saturating push occurs, respectively. Note that $|\Delta D| \le \lceil \log_2 U \rceil + 1$, $|R| \le 2n^2$ and $|SAT| \le nm$.

Let K be the last iteration. Each step $k \in \Delta D$, R, SAT or NS, so

$$\begin{aligned}\Phi(K) - \Phi(0) = &\sum_{k \in \Delta D} \Phi(k)-\Phi(k-1) + \sum_{k \in R} \Phi(k)-\Phi(k-1) \\ &\sum_{k \in SAT} \Phi(k)-\Phi(k-1) + \sum_{k \in NS} \Phi(k)-\Phi(k-1)\end{aligned}$$

Next we bound the relevant terms:

- $\Phi(0) \le n^2$ and $\Phi(K) = 0$
- if $k \in \Delta D$, then $\Phi(k)-\Phi(k-1) \le n^2$
- if $k \in R$, then $\Phi(k)-\Phi(k-1) \le$ increase in d(i)
- if $k \in SAT$, then $\Phi(k)-\Phi(k-1) \le 0$
- if $k \in NS$, then $\Phi(k)-\Phi(k-1) \le -1/2$

Thus $|NS|/2 \le n^2 + 2n^2 \log U + 2n^2 = O(n^2 \log U)$.

ADDITIONAL COMMENTS ON EXCESS SCALING

1. The algorithm can be modified (substantively) so that the running time is $O(nm + n^2 \log^{1/2} U)$.

2. The algorithm can be modified (a little) so that *any* node i with large excess may be selected for pushing, but if we try to push to a node j that has large excess, then we put i on a stack and try to push from j.

3. The algorithm works quite well in practice. (But highest level pushing is a little better.)

FURTHER RESULTS

1. Using the Dynamic Tree data structure, the running time of the pre-flow push algorithm can be reduced to $O(nm \log(n^2/m))$, but the algorithm is not very practical.

2. In the case of unit capacities, the max-flow problem can be solved in $O(n^{2/3} m)$ time. If at most one unit of flow can pass through each node, the running time is $O(n^{1/2} m)$.

3. In a bipartite network with $n = n_1 + n_2$ nodes, almost all pre-flow push algorithms can replace n by n_1 in the complexity.

4. In a planar network, the max-flow problem can be solved in $O(n^{3/2} \log n)$ time using planar separators.

5. The arc connectivity of a network (the number of arcs whose removal [strongly] disconnects the network) can be determined in $O(nm)$ time.

6. Hao and Orlin (1994) show that the overall minimum capacity cut in a network can be determined in time $O(nm \log(n^2/m))$.