Efficiently Maintaining the Edge List

**Discharge**$(v)$

1. while $e(v) > 0$ and (v has not been relabeled)
2.     $w = \text{current-edge}(v)$
3.     if $(v, w)$ is admissible
4.         $\text{push}(v, w)$
5.     elseif $(v, w)$ is not the last edge in $v$’s list
6.         $\text{Advance} \ \text{current-edge}(v)$
7.     else $\text{relabel}(v)$
8.     $\text{current-edge}(v) = \text{first-edge}(v)$
Observations

• In one call to discharge, all pushes except possibly the last one are saturating.
• Nonsaturating pushes cause discharge to terminate
• Each time the current edge point advances through the entire list $v$ is relabeled.

Conclusions

• Time spent advancing pointer is proportional to time spent relabeling.
• Data structure overhead = $O(nm)$. 
**FIFO Algorithm**

**Algorithm:** Keep active vertices in a queue. Call discharge from the vertex at the head of the queue, and add newly activated vertices to the rear of the queue.

**Phases**
- Phase 0: vertices added during initialization
- Phase $i$: vertices added during Phase $i-1$.

**Claim:** The number of phases is $O(n^2)$

**Proof:** Via potential function:

$$
\Phi = \max \{d(v) : e(v) > 0\}
$$

**FILL IN PROOF**
Bounding the number of non-saturating pushes

- At most 1 non-saturating push per Discharge
- At most $n$ Discharges per phase
- At most $O(n^2)$ phases

**Conclusion:** At most $O(n^3)$ non-saturating pushes.

Total run time =

\[ O(nm + n^3) = O(n^3) \]
Excess Scaling

Ideas:

- $e_{\text{max}} = \max_v \{ e(v) \}$.
- Only push flow from vertices with $e(v) \approx e_{\text{max}}$.
- Gradually decrease $e_{\text{max}}$.

Details:

- $\Delta$ is an upper bound on $e_{\text{max}}$.
- A node with $e(v) \geq \Delta/2$ has large excess.
- A node with $e(v) < \Delta/2$ has small excess.
Invariants to maintain

- Only push flow from nodes with large excess.
- Never let $e(v) > \Delta$ for any vertex.

Node selection rule: Choose, among vertices with large excess, the one with minimum distance label.

Implication: Any push goes from a vertex of large excess to one of small excess.
Excess Scaling Algorithm

1. Preprocess
2. \[ \Delta = 2^{\lceil \lg U \rceil} \]
3. while (\( \Delta \geq 1 \))
   4. while some node has large excess
      5. Let \( v \) be the min-dist label node of large excess
      6. \( w = \text{current-edge}(v) \)
      7. if \( \text{push}(v, w) \) is applicable
         8. push \( \delta = \min \{ e(v), c_f(v, w), \Delta - e(w) \} \) units of flow on \( (v, w) \)
      8. elseif \( (v, w) \) is not the last edge in \( v \)’s list
         9. \( \text{advance}(w) \)
      10. else \( \text{relabel}(v) \)
11. \( \Delta = \Delta / 2 \)
Lemma:

- Each non-saturating push send at least $\Delta/2$ units of flow
- No excess ever exceeds $\Delta$

Lemma: There are $O(n^2)$ non-saturating pushes per scaling phass.

Proof: Use potential function

$$\Phi = \sum_v e(v)d(v)/\Delta$$
Summary

- $O(n^2)$ relabels
- $O(nm)$ time relabeling
- $O(nm)$ saturating pushes
- $\lceil \lg U \rceil$ scaling phases, each does $O(n^2)$ non-saturating pushes.

Total Time: $O(nm + n^2 \lg U)$