Algorithm Theory

Chapter 6
Graph Algorithms

Part III:
Fast Ford Fulkerson Implementations

Fabian Kuhn

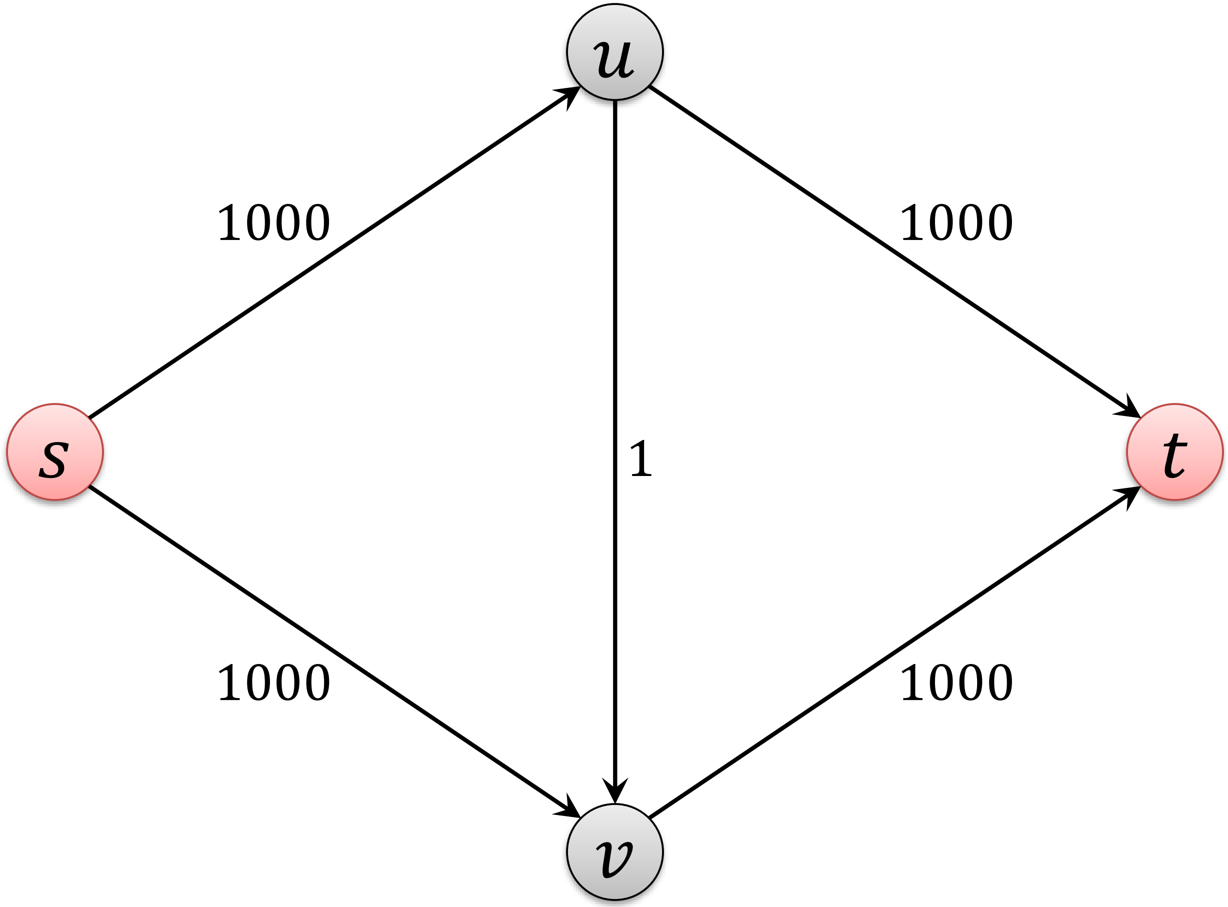
Non-Integer Capacities

If a given flow network has integer capacities, the Ford-Fulkerson algorithm computes a maximum flow of value C in time $O(m \cdot C)$.

What if capacities are not integers?

- rational capacities:
  - can be turned into integers by multiplying them with large enough integer
  - algorithm still works correctly
- real (non-rational) capacities:
  - not clear whether the algorithm always terminates
- even for integer capacities, time can linearly depend on the value of the maximum flow

Slow Execution



- Number of iterations: 2000 (value of max. flow)

Improved Algorithm

Idea: Find the best augmenting path in each step

- best: path P with maximum $\text{bottleneck}(P, f)$

- Best path might be rather expensive to find
  → find almost best path

- **Scaling parameter Δ:**
  (initially, $\Delta =$ "$\max c_e$ rounded down to next power of 2")

- As long as there is an augmenting path that improves the flow by at least Δ, augment using such a path

- If there is no such path: $\Delta \coloneqq {}^{\Delta}/_{2}$

Scaling Parameter Analysis

Lemma: If all capacities are integers, number of different scaling parameters used is $\leq 1 + \lfloor \log_2 c_{\max} \rfloor$.

$c_{\max} := \max_e c_e$

At the beginning: $\Delta = 2^{\lfloor \log_2 c_{\max} \rfloor}$
At the end: $\Delta = 1$

different scaling parameters Δ: $\lfloor \log_2 c_{\max} \rfloor + 1$

- **Δ-scaling phase:** Time during which scaling parameter is Δ

running time $=$ #scaling phases $\cdot$ #iterations per phase $\cdot\, O(m)$

#scaling phases: $O(\log c_{\max})$

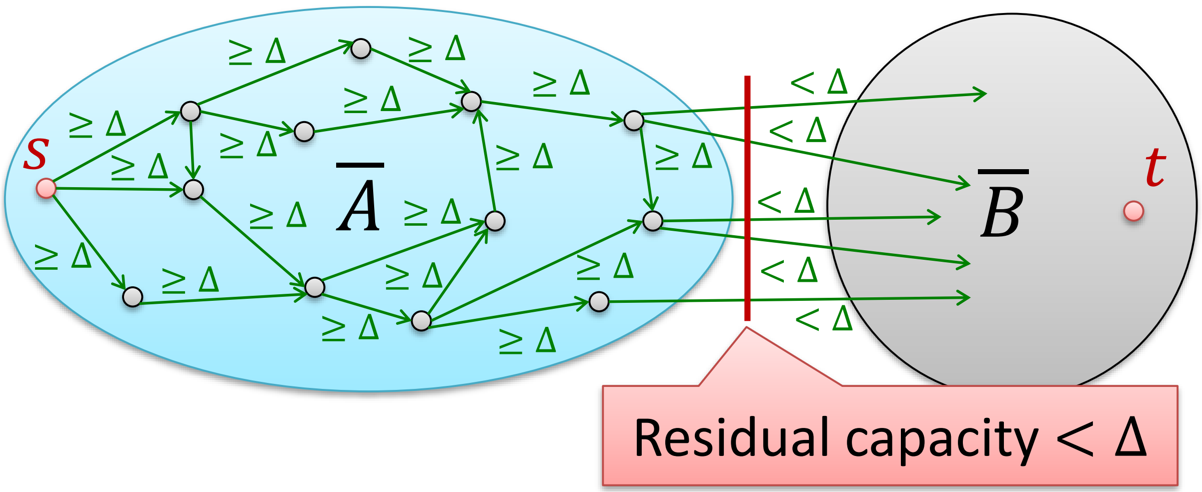
#iterations per phase: **???**

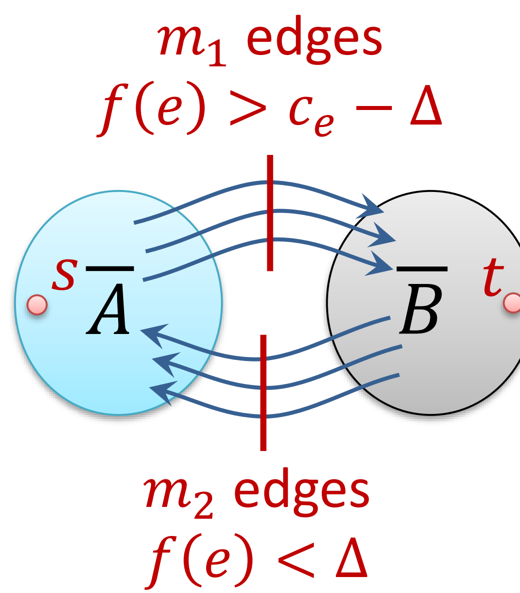
Length of a Scaling Phase

Lemma: If f is the flow at the end of the Δ-scaling phase, the maximum flow in the network has value less than $|f| + m\Delta$.

Proof:

- Define $\overline{A}$: set of nodes that can be reached from s on a path with residual capacities $\geq \Delta$ in G_f.





$$|f| = f^{\text{out}}(\overline{A}) - f^{\text{in}}(\overline{A}) > c(\overline{A}, \overline{B}) - m_1\Delta - m_2\Delta \geq c(\overline{A}, \overline{B}) - m\Delta$$

Length of a Scaling Phase

Lemma: The number of augmentation in each scaling phase is less than $2m$.

Proof:

- At the end of the 2Δ-scaling phase: $|f^*| < |f| + 2m\Delta$
- Each augmentation in the Δ-scaling phase improves the value of the flow f by at least Δ.
- #augmentations in Δ-scaling phase $< 2m$.

Running Time: Scaling Max Flow Alg.

Theorem: The number of augmentations of the algorithm with scaling parameter and integer capacities is at most $O(m \log c_{\max})$. The algorithm can be implemented in time $O(m^2 \log c_{\max})$.

Proof:

- #scaling phases: $O(\log c_{\max})$
- #iterations per scaling phase: $O(m)$
- time per iteration: $O(m)$

Strongly Polynomial Algorithm

- Time of regular Ford-Fulkerson algorithm with integer capacities:

$$O(mC)$$

- Time of algorithm with scaling parameter:

$$O(m^2 \log c_{\max})$$

- $O(\log c_{\max})$ is polynomial in the size of the input, but not in n
- Can we get an algorithm that runs in time polynomial in n?
- **Edmonds-Karp Alg.:** Always picking a shortest augmenting path:

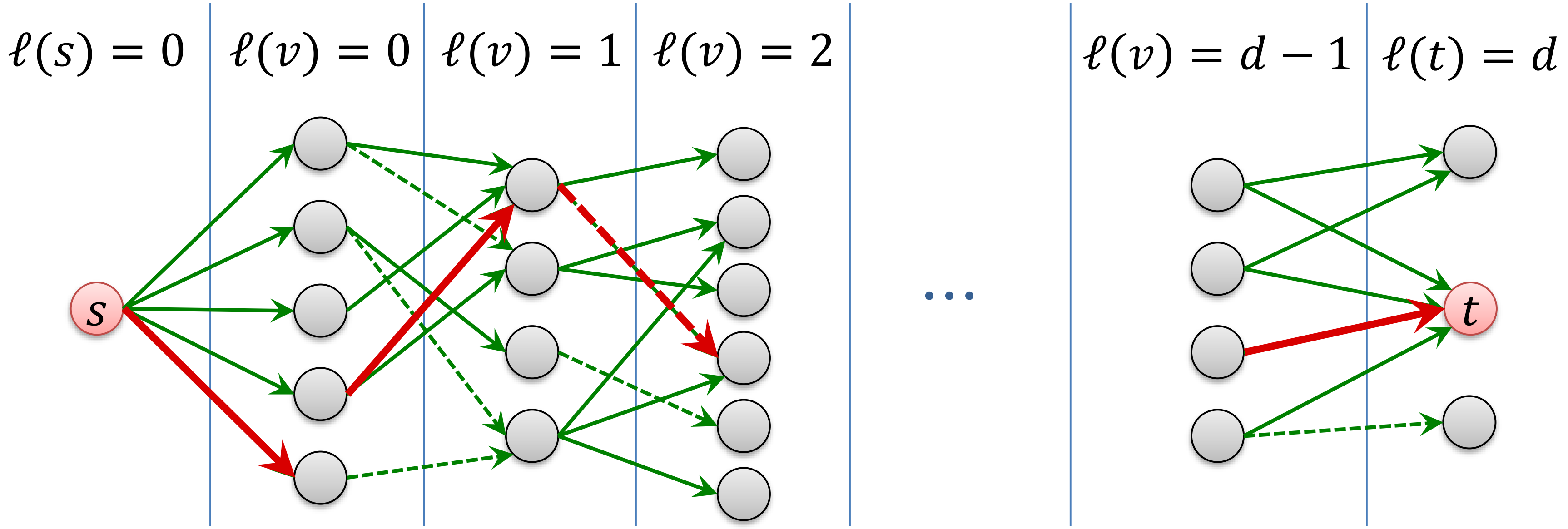
$$O(m^2 n)$$

  - also works for arbitrary real-valued weights
  - We will show this next.

Shortest Augmenting Path Algorithm

- Define G_f^+ as the subgraph of G_f with only the edges with positive residual capacity.
  - augmenting path = any s-t path in G_f^+

- **Level $\ell(v)$ of node v:**
  length (# of edges) of shortest path from s to v in G_f^+.

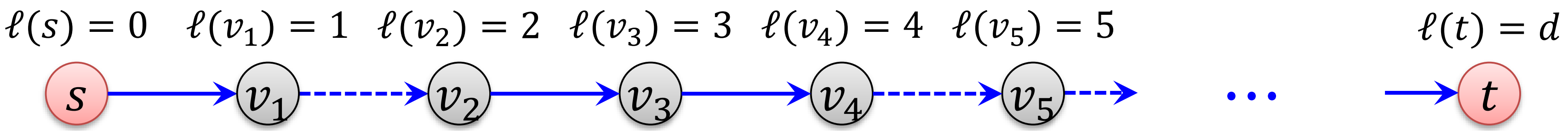


Shortest Augmenting Path Algorithm

Lemma 1: For every node v, the level $\ell(v)$ is non-decreasing.

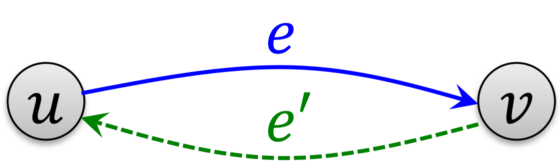
Proof:

- Consider augmentation along one augmenting path



  – Before augmentation, edges are between consecutive levels

- The set of edges of G_f^+ only changes if the residual capacity of some edge changes:



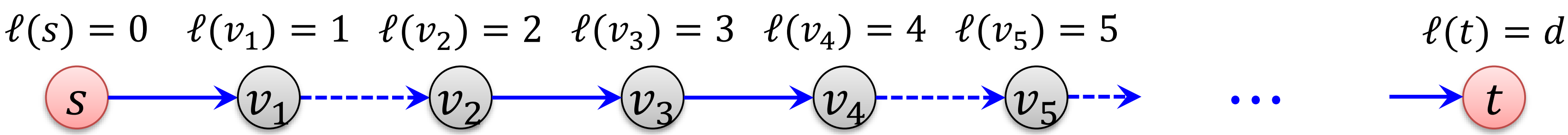
  – If e is on augmenting path P and $c_e = \text{bottleneck}(P, f)$, after augmentation, $c_e = 0$ and e is removed from G_f^+
  – The residual cap. of the edge e' in the opposite direction could increase from 0 to > 0 and be added to G_f^+

Shortest Augmenting Path Algorithm

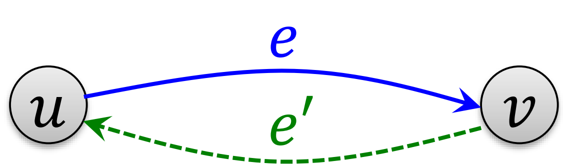
Lemma 1: For every node v, the level $\ell(v)$ is non-decreasing.

Proof:

- Consider augmentation along one augmenting path



  - Before augmentation, edges are between consecutive levels
  - A shortest augmenting path consists of exactly one node of each level.
  - The only new edges are from level $i+1$ to level i for some $i \geq 0$. (for the levels before augmenting along the path)



  - Such edges cannot create shortcuts to create s-w paths of length $< \ell(w)$
  - Levels of all nodes are non-decreasing.

Shortest Augmenting Path Algorithm

Lemma 2: There are at most $O(m \cdot n)$ augmentation steps.

Proof:

- In each augmentation step, at least one edge (u, v) is deleted from G_f^+
  - Some edge $e = (u, v)$ on the augmenting path P has $c_e = \text{bottleneck}(P, f)$
  - The residual capacity of e is set to 0 and e is removed from G_f^+
- When (u, v) is deleted from G_f^+, for some $i \geq 0$:

$$\ell(u) = i, \qquad \ell(v) = i + 1$$

- If (u, v) is later added back to G_f^+, for some $j \geq 0$:

$$\ell(u) = j + 1, \qquad \ell(v) = j$$

- Because level $\ell(v)$ is non-decreasing: $j \geq i + 1$
  $\Longrightarrow$ When (u, v) is added back, $\ell(u) \geq i + 2$
- Because the maximum possible level is $n - 1$, each edge is deleted from G_f^+ at most $O(n)$ times.

Shortest Augmenting Path Algorithm

Theorem: The Edmonds-Karp algorithm computes a maximum flow in time $O(m^2 n)$ even with arbitrary non-negative capacity values.

- *Edmonds-Karp algorithm = Ford-Fulkerson algorithm, where we choose a shortest augmenting path in each step.*

Proof:

- From lemma before: $O(m \cdot n)$ augmentation steps

- A shortest augmenting path can be found in time $O(m + n)$ by using a BFS traversal on the positive residual graph G_f^+.

Other Algorithms

- There are many other algorithms to solve the maximum flow problem, for example:

- **Preflow-push algorithm:** [Goldberg,Tarjan 1986]
  - Maintains a preflow (∀ nodes: inflow ≥ outflow)
  - Alg. guarantees: As soon as we have a flow, it is optimal
  - Detailed discussion in 2012/13 lecture
  - Running time of basic algorithm: $O(m \cdot n^2)$
  - Doing steps in the "right" order: $O(n^3)$

- **Current best known complexity: $\boldsymbol{O(m \cdot n)}$**
  - For graphs with $m \geq n^{1+\epsilon}$ (for every constant $\epsilon > 0$) [King,Rao,Tarjan 1992/1994]
  - For sparse graphs with $m \leq n^{16/15-\delta}$ [Orlin, 2013]