Notes on Minimum-Cost Flow Algorithms

Pramook Khungurn

December 11, 2019

1 Problem Definitions

- In the *minimum-cost flow problem*, you are given:
  - a graph $G = (V, E)$;
  - a *capacity function* $u : E \to \mathbb{R}^+ \cup \{0\}$;
  - a *demand function* (defined on the vertices) $b : V \to \mathbb{R}$;
  - a *cost function* $c : E \to \mathbb{R}$.

  You are to find a *flow* $f : E \to \mathbb{R}^+ \cup \{0\}$ such that
  - for all edge $e \in E$, we have $0 \le f(e) \le u(e)$;
  - for all vertex $v \in V$,
  $$\sum_{(v,w)\in E} f(v,w) - \sum_{(w,v)\in E} f(w,v) = b(v);$$
  - the *the cost of the flow* $\sum_{e\in E} c(e)f(e)$ is minimal.
- The min-cost flow problem is a generalization of many graphs problems. For examples:
  - The **shortest path** problem is a min-cost flow where you set (1) the capacity of every edge to 1, (2) the cost to its length, and (3) $b(s) = 1$ and $b(t) = -1$.
  - The **max flow** problem sets the cost of all edges to zero, and tries to find the maximum possible flow. (We shall see that this is actually equivalent to min-cost flow.)
  - The **disjoint path** problem asks to connect s and t with k disjoint paths using the smallest number of edges possible. This can be casted as a min-cost flow problem where (1) all the edges have capacity 1 and cost 1, and (2) $b(s) = k$ and $b(t) = -k$.
- In the *minimum-cost circulation problem*, you are given:
  - a graph $G = (V, E)$;
  - a *capacity function* $u : E \to \mathbb{R}^+ \cup \{0\}$;
  - a *demand function* $l : E \to \mathbb{R}^+ \cup \{0\}$ such that $l(e) \le u(e)$ for all $e \in E$;
  - a *cost function* $c : E \to \mathbb{R}$.

  You are to find a *circulation* $f : E \to \mathbb{R}^+ \cup \{0\}$ such that
  - for all edge $e \in E$, we have $l(e) \le f(e) \le u(e)$;
  - for all vertex $v \in V$,
  $$\sum_{(v,w)\in E} f(v,w) - \sum_{(w,v)\in E} f(w,v) = 0;$$

- the *the cost of the circulation* $\sum_{e\in E} c(e)f(e)$ is minimal.

- **Lemma 1.1.** *The min-cost flow problem and the min-cost circulation problem are equivalent.*

  Proof. (*circulation* $\implies$ *flow*) Let (G, u, b, c) be an instance of min-cost flow problem. Construct instance (G', u', l', c') of the min-cost circulation problem as follows:

  - Construct G' by adding a new vertex s to G.
  - For each $v \in V$ where $b(v) > 0$, add an edge (s, v) with $u'(s, v) = l'(s, v) = b(v)$, and $c(s, v) = 0$.
  - For each $v \in V$ where $b(v) < 0$, add an edge (v, s) with $u'(v, s) = l'(v, s) = |b(v)|$ and $c(v, s) = 0$.
  - For any other edge e, set $l'(e) = 0$ and $u'(e) = u(e)$ and $c'(e) = c(e)$.

  Then, there is a bijection between a feasible flow in G and a feasible circulation in G'. Moreover, since the new edges added to G' have cost 0, the cost of the flow equals to the cost of the circulation. Hence, a min-cost circulation gives a min-cost flow.

  (*flow* $\implies$ *circulation*) Let (G, u, l, c) be an instance of min-cost circulation problem. Construct instance (G', u', b', c') of the min-cost flow problem as follows:

  - G' is the same as G.
  - For all $e \in E$, set $u'(e) = u(e) - l(e)$.
  - For all $v \in V$, set $b'(v) = \sum_{(v,w)\in E} l(v, w) - \sum_{(w,v)\in E} l(w, v)$.
  - For all $e \in E$, set $c'(e) = c(e)$.

  There is a bijection between a feasible circulation in G and a feasible flow in G'. The cost of the flow in G' is not equal to the cost of circulation in G, but they differ by a constant. Hence, a min-cost flow gives a min-cost circulation. □

  Since the two problems are equivalent, we will work only with the min-cost circulation problem.

- The circulation problem's formulation can be simplified to make proofs and algorithm descriptions easier as follows:

  - We replace each edge by two edges of opposite direction.
  - For each newly created opposite (w, v) of edge (v, w), we set:
    * $u(w, v) = -l(v, w)$.
    * $c(w, v) = -c(v, w)$.
  - We eliminate the lower bound function altogether.

  Now, we define a new notion of circulation in the above graph as follows: a function $f : E \to \mathbb{R}$ is a *circulation* if the following conditions are satisfied:

  - $f(v, w) = -f(w, v)$ for all $(v, w) \in E$,
  - $f(e) \le u(e)$ for all $e \in E$, and
  - $\sum_{(v,w)\in E} f(v, w) = 0$.

  Let us note that the new formulation is equivalent to the old one. For each old edge (v, w), we have that $f(v, w) \ge l(v, w)$ iff $-f(v, w) \le -l(v, w)$. So the new opposite edges enforces the lower bound of the flow.

  If (v, w) is one of the edge in the graph of the previous formulation. The cost this edge incurs is $c(v, w)f(v, w)$. In the new formulation, there is also a flow of value $-f(v, w)$ going across (w, v). So the cost incurs by the edge (v, w) is actually

  $$c(v, w)f(v, w) + c(w, v)f(w, v) = c(v, w)f(v, w) + (-c(v, w)(-f(v, w)) = 2c(v, w)f(v, w).$$

  Hence, the cost of the circulation in the new formulation is two times that in the old formulation.

- **Lemma 1.2.** *Given one instance of the min-cost circulation problem, we can find whether there is a* feasible *solution by running a single max flow.*

  Proof. We revert the problem back to the version with demand. Let (G, u, l, c) be an instance of the min-cost circulation problem. For vertex v, define its *demand* $b(v)$ as $\sum_{(v,w)\in E} l(v,w) - \sum_{(w,v)\in E} l(w,v)$. We construct a new instance (G', u') for max flow problem as follows:

  - G' is obtained by adding a source vertex s and a sink vertex t to G.
  - For any edge (v, w) in G with positive capacity, we set $u'(v, w) = u(v, w) - l(v, w)$.
  - For any vertex v in G' with $b(v) > 0$, we create an edge (v, t) with $u'(v, t) = b(v)$.
  - For any vertex v in G' with $b(v) < 0$, we create an edge (s, v) with $u'(s, v) = -b(v)$.

  Note that it is the case that $\sum_{v\in V} b(v) = 0$. Let $B = \sum_{b(v)>0} b(v) = -\sum_{b(v)<0} b(v)$.
  It should be clear a max flow of value B yields a feasible circulation, and vice versa. □

2 Optimality Conditions

- From now on, let $G = (V, E)$ be a graph such that each directed edge has a backward edge. Let (G, u, c) be a flow network, and f be a circulation in it.
- The *residual network* (G_f, u_f, c) is given by:
  - $G_f = (V, E_f)$ where E_f is the set of edges such that $u(e) - f(e) > 0$.
  - $u_f(e) = u(e) - f(e)$ for all $e \in E_f$, and
- A function $p : V \to \mathbb{R}$ is called a *potential function* or a *price function*.
- Let p be a price function, and c be a cost function in network. The *reduced cost function* c^p is defined as: $c^p(v, w) = c(v, w) + c(v) - c(w)$.
- If Γ is a cycle in G and c is a cost function, let $c(\Gamma) = \sum_{e\in\Gamma} c(e)$.
- **Claim 2.1.** *For any cycle* Γ*, we have* $c(\Gamma) = c^p(\Gamma)$.

  Proof. Let $\Gamma = (v_1, v_2), (v_2, v_3), \ldots, (v_k, v_1)$. We have

$$\begin{aligned}
c^p(\Gamma) &= c^p(v_1, v_2) + c^p(v_2, v_3) + \cdots c(v_k, v_1) \\
&= [c(v_1, v_2) + p(v_1) - p(v_2)] + [c(v_2, v_3) + p(v_2) - p(v_3)] + \cdots + [c(v_k, v_1) + p(v_k) - p(v_1)] \\
&= [c(v_1, v_2) + c(v_2, v_3) + \cdots + c(v_k, v_1)] + [p(v_1) - p(v_2) + p(v_2) - p(v_3) + \cdots + p(v_k) - p(v_1)] \\
&= c(v_1, v_2) + c(v_2, v_3) + \cdots + c(v_k, v_1) = c(\Gamma).
\end{aligned}$$

- If c is a cost function and f a circulation, let $c \cdot f = \sum_{e\in E} c(e)f(e)$.
- **Claim 2.2.** *For any cost function c and price function p and circulation f, we have*

$$c \cdot f = c^p \cdot f.$$

Proof.

$$\begin{aligned}
c^p \cdot f &= \sum_{(v,w)\in E} c^p(v,w) f(v,w) \\
&= \sum_{(v,w)\in E} (c(v,w) + p(v) - p(w)) f(v,w) \\
&= \sum_{(v,w)\in E} c(v,w) + \sum_{(v,w)\in E} p(v) f(v,w) - \sum_{(v,w)\in E} p(w) f(v,w) \\
&= c \cdot f + \sum_{(v,w)\in E} p(v) f(v,w) - \sum_{(v,w)\in E} p(v) f(w,v) \\
&= c \cdot f + \sum_{(v,w)\in E} p(v)(f(v,w) - f(w,v)) \\
&= c \cdot f + \sum_{v\in V} p(v) \Big(\sum_{(v,w)\in E} f(v,w) - \sum_{(w,v)\in E} f(w,v) \Big) \\
&= c \cdot f.
\end{aligned}$$

The last equality is true because $\sum_{(v,w)\in E} f(v,w) = \sum_{(w,v)\in E} f(w,v) = 0$ because of flow conservation. □

- **Theorem 2.3.** *The following statements are equivalent:*

  (a) f is a minimum-cost circulation.

  (b) There is no negative-cost cycle in the residual network G_f.

  (c) There exists a price function p such that $c^p(e) \geq 0$ for all $e \in E_f$

  Proof. ($\neg(b) \to \neg(a)$) Augment along the negative-cost cycle gives a circulation with the lower cost.

  ($\neg(a) \to \neg(b)$) Let f be a feasible circulation that is not of minimum-cost. Let f^* be feasible circulation with minimum-cost. We have that $f * -f$ is a circulation that is feasible in G_f (because $f * (v,w) - f(v,w) \leq u(v,w) - f(v,w)$). Since f^* has lower cost than f, we have that $f^* - f$ has negative cost. We can decompose $f^* - f$ into cycles, and at least one cycle must be of negative cost.

  ($(b) \to (c)$) Start with G_f. Construct a new vertex s and connect s to every vertex v with $c(s,v) = 0$. Since there is no negative cycle, the shortest path distance is well-defined. Now, define $p(s) = 0$, and $p(v)$ = shortest path distance from s to v, taking c is the length of each edge.

  By property of shortest path distance, we have that $p(w) \leq p(v) + c(v,w)$ for all edge $(v,w) \in E_f$. Hence, $c^p(v,w) = c(v,w) + p(v) - p(w) \geq 0$.

  ($(c) \to (b)$) By Claim 2.2, $c(\Gamma) = c^p(\Gamma)$ for any cycle Γ. Since $c^p(e) \geq 0$ for all $e \in E$, we have that all cycle has positive cost. □

3 Cycle Canceling Algorithm

- The above theorem gives a simple algorithm for finding min-cost flow. Just find a negative-cost cycle and augment along it. Repeat until there are no negative-cost cycle.

  This algorithm is called Klein's algorithm.

- **Theorem 3.1.** *Let (G,u,c) be a network with integer capacity and integer cost. Let $U = \max_{e\in E}\{u(e)\}$ and $C = \max_{e\in E}\{|c(e)|\}$. Then, Klein's algorithm runs in $O(m^2 nUC)$.*

Proof. We can find a negative cycle in $O(mn)$ using Bellman–Ford algorithm. The minimum-cost is bounded below by $-mUC$. Each negative cycle decreases the cost of the circulation by at most -1. So mUC cycles suffice. □

4 Minimum Mean-Cost Cycle Canceling Algorithm

- For any cycle Γ, we define its *mean cost* $\mu(\Gamma) = c(\Gamma)/|\Gamma|$.
- Let μ^* be the minimum mean cost of all cycles. In other words, $\mu^* = \min_\Gamma \mu(\Gamma)$
- Instead of picking any negative-cost cycle, this algorithm picks the one with cost μ^* and augment along that cycle.

  Let $C = \max_{e \in E}\{|c(E)|\}$. It can be shown that only $O(m^2 n \log n)$ augmentation suffices. We will not be showing why this is true.
- An interesting problem is how to find the minimum mean cost cycle. We will present two algorithms: one with complexity $O(mn \log(n^2 C))$ and the other with complexity $O(mn)$.
- In the $O(mn \log(n^2 C))$ algorithm, we assume that the edges have integer costs. The idea is to binary search for μ^*. We know that $\mu^* \in [-C, C]$ so we can set the search interval accordingly.

  Suppose that we guess $\mu^* = a$. We will subtract a from all the edge cost of the graph. Notice that for any cycle, its mean cost is reduced by a. We have the following case.

  - If $\mu^* \geq a$, then all the cycles have positive mean cost, and thus they have positive cost.
  - If $\mu^* = a$, then there exists some cycle with zero cost, but none of the cycles have negative cost.
  - If $\mu^* < a$, then some cycles have negative cost.

  We can check whether there is a negative-cost cycle by running Bellman–Ford algorithm, which takes $O(mn)$ time.

  How many iterations do we need? Since the denominator of μ^* is an integer from 1 to n, we have that two candidate values for μ^* cannot differ by more than $1/n^2$. So, when the interval is smaller than $1/n^2$, we can be sure that only one candidate is inside. Hence, we need at most $O(\log(n^2 C))$ iterations to shrink the interval to this size.

  Once the interval is small enough, we can find the value by searching though all possible denominators, which takes $O(n)$ time. Overall, the algorithm takes $O(mn \log(n^2 C))$ time.
- Once we find μ^*, how can we find a cycle with minimum mean cost?

  We first subtract μ^* form all edge cost. Since there is no negative-cost cycle, by Theorem 2.3 there exists a price function p such that $c^p(e) \geq 0$ for all $e \in E$. This function can be founded by taking $p(v) =$ the shortest path distance from a new vertex s with an edge of cost 0 pointing to every vertex.

  Now, the cycle with minimum mean cost turns into a cycle with zero cost. Since all edges have non-negative cost, the cycle has to consists only of edges with zero cost. We can locate all those edges and find a cycle formed by them.
- The $O(mn)$ algorithm is rather involved. We start with a definition.

  Definition 4.1. *Let v be a vertex and k be a non-negative integer. Define $d_k(v)$ to be the cost of a walk containing exactly k edges ending at v with the least possible cost.*

  Then, μ^* can be characterized as follows:

Lemma 4.2.

$$\mu^* = \min_{v \in V} \max_{0 \le k \le n-1} \left\{ \frac{d_n(v) - d_k(v)}{n-k} \right\}$$

Proof. Let α denote the RHS of the equation. Notice that if we subtract a from all the edge cost then α is reduced by a as well. Hence, we can show that $\mu^* = \alpha$ by working in a graph where μ^* is subtracted from the cost of all edge and show that $\alpha = 0$ in this graph. Note that, in this graph, all cycles have positive cost, and there is at least one cycle with zero cost.

($\alpha \ge 0$) Let

$$\alpha(v) = \max_{0 \le k \le n-1} \frac{d_n(v) - d_k(v)}{n-k}.$$

Let v be the vertex where $\alpha(v)$ is minimum. Let p be the walk of length n ending at v with minimum cost, i.e., $c(p) = d_n(v)$. Since p has length n, it must contain a cycle. Thus, we can compose p into a cycle π and a path τ leading to v. Let j be the number of edges in τ. We must have that $c(\tau) = d_j(v)$, otherwise p would not have been the shortest walk. So,

$$\alpha(v) = \max_{0 \le k \le n-1} \frac{d_n(v) - d_k(v)}{n-k} \ge \frac{d_n(v) - d_j(v)}{n-j} = \frac{c(\tau)}{n-j} \ge 0.$$

($\alpha \le 0$) Let Γ be a cycle with cost 0, and let v be a vertex in the cycle.

Consider the sequence $d_0(v), d_1(v), d_2(v), \ldots$. We claim that there exists r such that $d_r(v)$ is minimum and $r < n$. That is, Suppose that $r \ge n$. Since the walk that achieves $d_r(v)$ has at least n edges, it must contain a cycle. We can take the cycle out without increasing $d_r(v)$ until there $r < n$.

Let η be a walk from v that proceeds along the cycle for $n - r$ hops, and let the last vertex in the hop be w. Let τ be the walk from w along the cycle to v. We have that $c(\tau) + c(\eta) = 0$. So, for any k, we have that

$$d_k(w) = d_v(w) + c(\tau) + c(\eta) \ge d_r(v) + c(\eta) \ge d_n(w).$$

The inequality $d_v(w) + d(\tau) \ge d_r(v)$ comes from the fact that $d_v(w) + d(\eta)$ is the length of a path to v. Moreover, $d_r(v) + c(\eta) \ge d_n(v)$ because $d_r(v) + c(\eta)$ is the length of a path of length n to w. Hence, $d_n(w) - d_k(w) \le 0$. So, $\alpha \le 0$. □

- The $O(mn)$ algorithm computes $d_k(v)$ for all $0 \le k \le n$ and $v \in V$, which can be done by dynamic programming in $O(mn)$. It then compute μ^* according to the formula given by Lemma 4.2, and then uses μ^* to find the minimum mean cost cycle.

5 Cost Scaling Algorithm

- A potential function is said to be ϵ-*optimal* if $c^p(e) \ge -\epsilon$ for all $e \in E$.
- Let f be a circulation. Define $\epsilon(f)$ to be the minimal ϵ such that there exists a potential function that is ϵ-optimal in G_f.
- Let $\mu^*(f)$ be the mean cost of the cycle in G_f with the minimum cost.
- **Lemma 5.1.**

$$\mu^*(f) = -\epsilon(f)$$

Proof. ($\mu^*(f) \leq -\epsilon(f)$) Subtract $\mu^*(f)$ from all edge cost in the residual graph. Add a new vertex s and add edge (s, v) with cost 0 to all $v \in V$. Define $p(v)$ to be the shortest path distance from s to v. We know that $p(w) \leq p(v) + c(v, w) - \mu^*(f)$ for all edges in G_f. Hence, $c(u, v) + p(v) - p(w) \geq \mu^*(f)$. Therefore, p is $-\mu^*(f)$-optimal, which means that $\epsilon(f) \leq -\mu^*(f)$ or $\mu^*(f) \leq -\epsilon(f)$.

($\mu^*(f) \geq -\epsilon(f)$) Let p be a position function that is $\epsilon(f)$-optimal. Take any cycle Γ with the minimum negative mean cost in G_f. We have that $|\Gamma|\mu^*(f) = c(\Gamma) = c^p(\Gamma) \geq -|\Gamma|\epsilon(f)$. Therefore, $\mu^*(f) \geq -\epsilon(f)$. ☐

- **Lemma 5.2.** *In a network with integer cost, if $\epsilon(f) < 1/n$, then f is the min-cost circulation.*

  Proof. Let p be a potential function that is $\epsilon(f)$ optimal. Take any cycle Γ of length at most n in G_f. We have that $c(\Gamma) = c^p(\Gamma) \geq |\Gamma|\mu^*(f) = -|\Gamma|\epsilon(f) > -|\Gamma|/n = 1$. Since the cost of the cycle is integral, we have that $c(\Gamma) \geq 0$.

  Now, for any cycle of length more than n, we can break it to a number of cycles of length at most n. So, its cost is greater than or equal to 0 as well. In conclusion, there is no negative-cost cycle, and f is optimal. ☐

- The idea of cost-scaling algorithm is that, when we start with $f = 0$, we have that $\epsilon(f) \leq C$. We will then do something to the flow so that $\epsilon(f)$ is reduced by a half until it is less than $1/n$, which at thas point we have an optimal flow. Hence, we will need $\log(nC)$ iterations.

- A function $f : E \to \mathbb{R}$ is said to be a *preflow* if it satisfies the following condition:
  - $f(v, w) = -f(w, v)$, and
  - $f(v, w) \leq u(v, w)$

  for all $(v, w) \in E$.

- Let f be a preflow. We define the *excess* of vertex v, denoted by $e^f(v)$ as

  $$e^f(v) = \sum_{(v,w)\in E} f(v, w).$$

  Intuitively, a vertex with positive excess has left-over flow to send out. One with negative excess needs flow to come in.

- Obviously, if all vertices' excesses are zero, then the preflow is a circulation.

- We now describe an algorithm that takes in
  - a circulation f', and
  - a potential function p' that is 2ϵ-optimal in $G_{f'}$

  and produces a
  - a circulation f, and
  - a potential function p that is ϵ-optimal in G_f.

  We first set $f = f'$ and $p = p'$. We then make p 0-optimal by saturating any edges with $c^p(e) < 0$, thereby removing them from G_f. However, this makes f a preflow, not a circulation.

  The rest of the process is to make f a circulation again, while maintaining ϵ-optimality of p. This is done by a push/relabel type of algorithm as follows:

– **Push:** If there is an edge (v, w) such that $e^f(v) > 0$ and $u^f(v, w) > 0$ and $c^p(v, w) < 0$, we push $\min\{e^f(v), u^f(v, w)\}$ through the edge.
– **Relabel:** For any vertex v with no edges that flow can be pushed trough, we set

$$p(v) = \max_{(v,w)\in E}\{p(w) - c(v, w) - \epsilon.\}$$

Note that setting $p(v)$ to this value makes $c^p(v, w) = p(v) - p(w) + c(v, w) \geq -\epsilon$ for all edges going out from v.

The above description can be summarized into the following pseudocode.

FIND-ϵ-OPT-CIRC(f', p', ϵ)

Set $f = f'$ and $p = p'$.
For all $e \in E_f$ such that $c^p(e) < 0$, set $f(e) = u(e)$.
while there exists v such that $e^f(v) > 0$
    if there exists (v, w) such that $u^f(u, v) > 0$ and $c^p(u, v) < 0$
        Push flow $\min\{u^f(u, v), e^f(v)\}$ through (u, v).
    else Set $p(v) = \max_{(v,w)\in E}\{p(w) - c(v, w) - \epsilon\}$.

Using the FIFO implementation of Push/Relabel algorithm, the routine FIND-ϵ-OPT-CIRC(f', p', ϵ) runs in $O(n^3)$ time.

- Hence, the cost scaling algorithm takes $O(n^3 \log(nC))$ time.

6 Capacity Scaling Algorithm

- The capacity scaling algorithm tries to maintain a preflow that is 0-optimal on subgraphs with residual capacity more than Δ. It then divides Δ until $\Delta < 1$, at which point the preflow becomes the optimal circulation.

  Initially, we set $f = 0$, $p = 0$, and $\Delta = U = \max_{e\in E} u(e)$.

  Let
  $$A^f(\Delta) = \{e \in E : u^f(e) \geq \Delta\}.$$

  The algorithm looks for any edge $e \in A^f(\Delta)$ such that $c^p(e) < 0$. It then pushes Δ amount of flow through the edge. This creates a preflow where some nodes have positive excesses and some have negative excesses. Let

  – $S^f(\Delta) = \{v \in V : e^f(v) \geq \Delta\}$, and
  – $T^f(\Delta) = \{v \in V : e^f(v) \leq -\Delta\}$.

  The algorithm then tries to move Δ unit of flow from a vertex in $S^f(\Delta)$ to one in $T^f(\Delta)$ until $S^f(\Delta) = \emptyset$ or $T^f(\Delta) = \emptyset$, while maintaining 0-optimality of p. It then divides Δ by 2 and proceed with the above process again.

  Note that it must be possible to move flow from a vertex to any other vertex in other for the algorithm to work. This can be made possible by adding edges with infinite capacity but with high cost to the graph.

- **Lemma 6.1.** *In a network with integer capacity, when $\Delta < 1$, then f is an optimal circulation.*

  Proof. After the algorithm finishes $S^f(\Delta) = \{v \in V : e^f(v) \geq 1\} = \emptyset$ and $T^f(\Delta) = \{v \in V : e^f(v) \leq -1\} = \emptyset$. This implies that all $e^f(v) = 0$. So f is a circulation.

  When $\Delta < 1$, then $A^f(\Delta)$ is the whole graph G_f. Since p is 0-optimal, we have that G_f has no negative-cost cycle. ☐

- How do we send Δ unit of flow from a vertex in $S^f(\Delta)$ to a vertex in $T^f(\Delta)$ while maintaining 0-optimality of p? We do this as follows:
  - Find a vertex $s \in S^f(\Delta)$.
  - Compute $\hat{p}(v) =$ shortest path distance from s to v using $c^p(e)$ as length of edge e. This can be done because p is 0-optimal.
  - Compute a shortest path from s to some vertex in $T^f(e)$. Push Δ unit of flow along the shortest path.
  - Update $p(v) = p(v) + \hat{p}(v)$.

  This procedure amounts to running Dijkstra's algorithm one time. So it takes $O(m + n \log n)$ time.

- It can be shown that the total amount of excess after saturating edges is at most $2\Delta(n + m)$. Hence, $O(m + n)$ flow pushing is enough before we halve Δ. Therefore, the whole algorithm takes $O((m + n)(m + n \log n) \log U) = O(m^2 \log U)$ time.