1 The Min-Cost Flow Problem

In many applications, there are many flows of maximum value. For example, think about a network with two vertex-disjoint s-t paths of equal capacity, but one path is much longer than the other. In general, one can think about *edge costs*: sending a flow of one unit over an edge e, we need to pay a price $p(e)$. The new objective is thus to find a maximum flow that minimizes the total cost. This is called the *minimum-cost flow problem.*

Instead of s-t-networks, we will consider general networks, where each vertex v has a supply or demand $b(v)$. One can think of s-t-networks as a special case, where $b(s) = k$, $b(t) = -k$, and $b(v) = 0$ for all other vertices v, and k the value of a maximum flow. In fact, one can also reduce finding a feasible flow in a general network to finding a maximum flow in an s-t-network. We leave this argument as an exercise.

We start with the following definitions.

Definition 1 (Network with costs)**.** *A tuple $\mathcal{N} = (V, E, c, b, p)$ is a* network *with cost if*

- *V is a finite set of vertices,*
- *$E \subseteq V \times V$ is a set of directed edges,*
- *$c : E \to \mathbb{R}_{\geq 0}$ capacities,*
- *$b : V \to \mathbb{R}$ is a supply function, and*
- *$p : E \to \mathbb{R}_{\geq 0}$ is a cost function.*

Definition 2 (Feasible flow)**.** *A feasible* flow *in a network $\mathcal{N} = (V, E, c, b, p)$ is a function $f : E \to \mathbb{R}_{\geq 0}$ such that*

- *$f(e) \leq c(e)$ for all $e \in E$,*
- *$\sum_{e\in\delta^+(v)} f(e) - \sum_{e\in\delta^-(v)} f(e) = b(v)$ for all $v \in V$.*

Definition 3 (Cost of a flow)**.** *The* cost *of a flow f in a network $\mathcal{N} = (V, E, c, b, p)$ is defined as*

$$\mathrm{cost}(f) = \sum_{e\in E} p(e)f(e).$$

Definition 4 (Circulation)**.** *A* circulation *f in a network $\mathcal{N} = (V, E, c, b, p)$ is flow f that is feasible for the network $(V, E, c, 0, p)$.*

2 The Cycle-Augmenting Algorithm

We start with an algorithm that can be seen as the analogue of the generic path-augmenting algorithm by Ford and Fulkerson for the min-cost flow problem. The main difference is that instead of searching augmenting *paths* in $\mathcal{N}_f$, we seek for cycles in $\mathcal{N}$ with negative cost, as they intuitively can reduce the overall cost of the current flow.

Formally, the *cost of a cycle C* in a network $\mathcal{N} = (V, E, c, b, p)$ is defined as the sum of the costs of the edges in C: $\mathrm{cost}(C) = \sum_{e\in C} p(e)$. We call a cycle C *negative* if $\mathrm{cost}(C) < 0$.

Lemma 1. *Let f and f' be two feasible flows in a network $\mathcal{N} = (V, E, c, b, p)$. Then, $f' - f$ is a circulation in $\mathcal{N}_f$.*

Proof. As an exercise. □

Lemma 2. *Let f be a circulation in a network $\mathcal{N} = (V, E, c, b, p)$. Then, there exist flows $f_1, \ldots, f_k$ with $k \leq m$ such that*

(i) $f = f_1 + \ldots + f_k$,

(ii) f_i *is a feasible flow in* $\mathcal{N}$ *for all* $i \in [k]$, *and*

(iii) f_i *takes positive values only on edges of a cycle* C_i *in* $\mathcal{N}$ *for all* $i \in [k]$.

Proof. As an exercise. □

The following lemma is an optimality criterion for min-cost flows.

Lemma 3. *A feasible flow f in a network $\mathcal{N} = (V, E, c, b, p)$ is optimal if and only if there is no negative cycle in $\mathcal{N}_f$.*

Proof. If there is a negative cycle C in $\mathcal{N}_f$ and f is optimal, we can augment f along C to obtain a new feasible flow f' with $\text{cost}(f') < \text{cost}(f)$, contradicting the optimality of f.

For the other direction, let f' be a feasible flow in $\mathcal{N}$. By Lemma 1, $f - f'$ is a circulation in $\mathcal{N}$. Moreover, by Lemma 2, we can write $f - f' = f_1 + \ldots + f_k$ such that f_i is a feasible flow in $\mathcal{N}_f$ for all $i \in [k]$ and f_i takes positive values only on edges of a cycle C_i in $\mathcal{N}_f$. Since f contains no negative cycles, we have $\text{cost}(f') = \text{cost}(f) + \sum_{i=1}^{k} \text{cost}(f_i) \geq \text{cost}(f)$, which shows that f is optimal. □

We can augment a flow f along a cycle C in $\mathcal{N}_f$ by increasing the flow on the edges of C by the minimum capacity of the edges in C, similarly to augmenting a flow along a s-t-path in an s-t-network. Formally, we have a bottleneck capacity of $\gamma = \min_{e \in C} c_f(e)$, and update

$$f(e) \leftarrow \begin{cases} f(e) + \gamma & \text{if } e \in C \cap E \\ f(e) - \gamma & \text{if } e \in C \cap \overleftarrow{E} \\ f(e) & \text{otherwise} \end{cases}$$

for all $e \in E \cup \overleftarrow{E}$.

Whenever we augment a flow along a negative cycle, we decrease the total cost of the flow. This suggests a generic algorithm (Algorithm 2), which by Lemma 3 computes a min-cost flow.

Algorithm 1: Generic Cycle-Augmenting Algorithm

Input: A network $\mathcal{N} = (V, E, c, b, p)$
Output: An min-cost flow f in $\mathcal{N}$

```
Let f be any feasible flow in N
while there is a negative cycle C in N_f do
    Augment f along C
return f
```

Theorem 1. *Let $\mathcal{N} = (V, E, c, b, p)$ be a network with costs. If the values $c(e)$, $b(v)$, and $p(e)$ are integers for all $e \in E$ and $v \in V$, then the generic Cycle-Augmenting algorithm computes a min-cost flow in $\mathcal{N}$ in time $O(m^2 n \cdot C \cdot P)$, where n is the number of vertices, m is the number of edges, C is the maximum capacity of an edge in $\mathcal{N}$, and P is the maximum cost of an edge in $\mathcal{N}$.*

Proof. From the above discussion and Lemma 3, it follows that the algorithm is correct and computes a min-cost flow.

It remains to bound the running time. In the previous lectures, we have seen that we can compute an initial feasible flow in time $O(n^2m)$. Finding a negative cycle in a graph can be done with the algorithm of Bellman and Ford in time $O(nm)$. By our integrality assumption, each iteration decreases the cost by at least 1. The total cost of any flow lies between $-mCP$ and mCP, the number of iterations is bounded by $2mCP$. In total, the running time is in $O(m^2n \cdot C \cdot P)$. $\square$

Since in an integer network with cost, we can compute a feasible integral flow initially, each iteration preserves the integrality, and we can conclude the following corollary.

Corollary 1. *Let $\mathcal{N} = (V, E, c, b, p)$ be a network with costs. If the values $c(e)$, $b(v)$, and $p(e)$ are integers for all $e \in E$ and $v \in V$, then there exists a min-cost flow f in $\mathcal{N}$ such that $f(e)$ is an integer for all $e \in E$.*

3 The Minimum-Mean Cycle-Augmenting Algorithm

The running time of the generic Cycle-Augmenting Algorithm is not very good. In particular, it depends linearly on C and P, and thus, is only *pseudopolynomial.* In this section, we study a particular implementation of the Cycle-Augmenting algorithm, which runs in strongly polynomial time (independent of C and P).

Intuitively, we want to find a negative cycle that reduces the cost as much as possible; that is, a cycle with of minimum cost. Unfortunately, the problem of finding such a cycle is NP-hard. Luckily, we can find a cycle with minimum *mean* cost in polynomial time, which is defined as the cost of the cycle divided by the number of edges in the cycle. This turns out to be a good-enough approximation for the minimum cost cycle: one can show that augmenting along such cycles leads to a strongly polynomial algorithm.

Algorithm 2: Minimum-Mean Cycle-Augmenting Algorithm

Input: A network $\mathcal{N} = (V, E, c, b, p)$
Output: An min-cost flow f in $\mathcal{N}$

Let f be any feasible flow in $\mathcal{N}$
while *there is a negative cycle in* $\mathcal{N}_f$ **do**
    Let C be a negative cycle in $\mathcal{N}_f$ of minimum-mean cost
    Augment f along C
return f

Theorem 2. *The Minimum-Mean Cycle-Augmenting Algorithm computes a min-cost flow in an integral network with costs with n vertices and m edges in time $O(n^2m^3 \log n)$.*