Network flows

1.1 Maximum $s-t$-flows

We now turn our attention to a linear programming problem which we will solve by direct methods, motivated by the nature of the problem. We often use the following notation. If $f : A \longrightarrow B$ denotes a function and if $U \subseteq A$, then $f(U)$ is defined as $f(U) = \sum_{a \in U} f(a)$.

Definition 19 (Network, $s-t$-flow)**.** A network with capacities consists of a directed simple graph $D = (V, A)$ and a *capacity function* $u : A \to \mathbb{R}_{\geqslant 0}$. A function $f : A \to \mathbb{R}_{\geqslant 0}$ is called an $s-t$*-flow*, if

$$\sum_{e \in \delta^{out}(v)} f(e) = \sum_{e \in \delta^{in}(v)} f(e), \text{ for all } v \in V - \{s, t\}, \tag{1.12}$$

where $s, t \in V$. The flow is *feasible*, if $f(e) \leqslant u(e)$ for all $e \in A$. The *value* of f is defined as $value(f) = \sum_{e \in \delta^{out}(s)} f(e) - \sum_{e \in \delta^{in}(s)} f(e)$. The *maximum* $s-t$*-flow problem* is the problem of determining a maximum feasible $s-t$-flow.

Here, for $U \subseteq V$, $\delta^{in}(U)$ denotes the arcs which are entering U and $\delta^{out}(U)$ denotes the arcs which are leaving U. Edge sets of the form $\delta^{out}(U)$ are called a *cut* of D. The *capacity of a cut* $c(\delta^{out}(U))$ is the sum of the capacities of its edges.

Thus the maximum flows problem is a linear program of the form

$$\max \sum_{e \in \delta^{out}(s)} x(e) - \sum_{e \in \delta^{in}(s)} x(e) \tag{1.13}$$

$$\sum_{e \in \delta^{out}(v)} x(e) = \sum_{e \in \delta^{in}(v)} x(e), \text{ for all } v \in V - \{s, t\} \tag{1.14}$$

$$x(e) \leqslant u(e), \text{ for all } e \in A \tag{1.15}$$

$$x(e) \geqslant 0, \text{ for all } e \in A \tag{1.16}$$

$$\tag{1.17}$$

Definition 20 (excess function)**.** For any $f : A \to \mathbb{R}$, the excess function is the function $excess_f : 2^V \to \mathbb{R}$ defined by $excess_f(U) = \sum_{e \in \delta^{in}(U)} f(e) - \sum_{e \in \delta^{out}(U)} f(e)$.

Theorem 21. *Let $D = (V, A)$ be a digraph, let $f : A \to \mathbb{R}$ and let $U \subseteq V$, then*

$$excess_f(U) = \sum_{v \in U} excess_f(v). \tag{1.18}$$

Proof. An arc which has both endpoints in U is counted twice with different parities on the right, and thus cancels out. An arc which has his tail in U is subtracted once on the right and once on the left. An arc which has his head in U is added once on the right and once on the left. □

A cut $\delta^{out}(U)$ with $s \in U$ and $t \notin U$ is called an $s - t$-cut.

Theorem 22 (Weak duality)**.** *Let f be a feasible $s - t$-flow and let $\delta^{out}(U)$ be an $s - t$-cut, then $value(f) \leqslant u(\delta^{out}(U))$.*

Proof. $value(f) = -excess_f(s) = -excess_f(U) = f(\delta^{out}(U)) - f(\delta^{in}(U)) \leqslant f(\delta^{out}(U)) \leqslant u(\delta^{out}(U))$. □

For an arc $a = (u, v) \in A$ the arc a^{-1} denotes the arc (v, u).

Definition 23 (Residual graph)**.** Let $f : A \to \mathbb{R}$, $l : A \to \mathbb{R}$ and $u : A \to \mathbb{R}$ where $l \leqslant f \leqslant u$. Consider the sets of edges

$$A_f = \{a \mid a \in A,\ f(a) < u(a)\} \cup \{a^{-1} \mid a \in A,\ f(a) > l(a)\}. \tag{1.19}$$

The digraph $D(f) = (V, A_f)$ is called the *residual graph* of f (for lower bound l and capacities u).

Definition 24 (walk, path, distance)**.** A *walk* is a sequence of the form $P = (v_0, a_1, v_1, \ldots, v_{m-1}, a_m, v_m)$, where $a_i = (v_{i-1}, v_i) \in A$ for $i = 1, \ldots, m$. If the nodes $v_0, \ldots, v_m$ are all different, then P is a *path*. The *length* of P is m. The *distance* of two nodes u and v is the length of a shortest path from u to v.

Corollary 25. *Let f be a feasible $s - t$-flow and suppose that $D(f)$ has no path from s to t, then f has maximum value.*

Proof. Let U be the set of nodes which are reachable in $D(f)$ from s. Clearly $\delta(U)$ is an $s-t$-cut. Now $value(f) = f(\delta^{out}(U)) - f(\delta^{in}(U)$. Each arc leaving U is not an arc of $D(f)$ and thus $f(\delta^{out}(U)) = u(\delta^{out}(U))$. Each arc entering U is not an arc of $D(f)$ and thus $f(\delta^{in}(U) = 0$. It follows that $value(f) = u(\delta^{out}(U))$ and f is optimal by Theorem 22. □

Definition 26 (undirected walk)**.** An undirected walk is a sequence of the form $P = (v_0, a_1, v_1, \ldots, v_{m-1}, a_m, v_m)$, where $a_i =\in A$ for $i = 1, \ldots, m$ and $a_i = (v_{i-1}, v_i)$ or $a_i = (v_i, v_{i-1})$. If the nodes $v_0, \ldots, v_m$ are all different, then P is a *path*. The *length* of P is m. The *distance* of two nodes u and v is the length of a shortest path from u to v.

Any directed path P in $D(f)$ yields an undirected path in D. Define for such a path P $\chi^P \in \{0, \pm 1\}^A$ as

$$\chi^P(a) = \begin{cases} 1 & \text{if } P \text{ traverses } a, \\ -1 & \text{if } P \text{ traverses } a^{-1}, \\ 0 & \text{if } P \text{ traverses neither } a \text{ or } a^{-1}. \end{cases} \tag{1.20}$$

Theorem 27 (max-flow min-cut theorem, strong duality)**.** *The maximum value of a feasible $s-t$-flow is equal to the minimum capacity of an $s-t$ cut.*

Proof. Let f be a maximum $s-t$-flow. Consider the residual graph $D(f)$ with lower bound $l=0$. If this residual graph contains an $s-t$-path P, then we can route flow along this path. More precisely, there exists an $\epsilon > 0$ such that $f + \epsilon\,\chi^P$ is feasible. We have $value(f + \epsilon\,\chi^P) = value(f) + \epsilon$. This contradics the maximality of f thus there exists no $s-t$-path in $D(f)$.

Let U be the nodes reachable from s in $D(f)$. Then $value(f) = u(\delta^{out}(U))$. □

This suggests the algorithm of Ford and Fulkerson to find a maximum flow. Start with $f=0$. Next iteratively apply the following *flow augmentation algorithm*.

Let P be a directed $s-t$-path in $D(f)$. Set $f \leftarrow f + \epsilon\chi^P$, where ϵ is as large as possible to maintain $0 \leqslant f \leqslant u$.

Exercise 5. Define a *residual capacity* for $D(f)$. Then determine the maximum ϵ such that $0 \leqslant f \leqslant u$.

Theorem 28. *If all capacities are rational, this algorithm terminates.*

Exercise 6. Provide a proof of Theorem 28.

Here a picture with diamond needing exponentially many augmentations. Explain!

Corollary 29 (integrity theorem)**.** *If $u(a) \in \mathbb{N}$ for each $a \in A$, then there exists an integer maximum flow ($f(a) \in \mathbb{N}$ for all $a \in A$).*

See [8]

Theorem 30. *If we choose in each iteration a shortest $s-t$-path in $D(f)$ as a flow-augmenting path, the number of iterations is at most $|V| \cdot |A|$.*

Definition 31. Let $D = (V, A)$ be a digraph, $s, t \in V$ and let $\mu(D)$ denote the length of a shortest path from s to t. Let $\alpha(D)$ denote the set of arcs contained in at least one shortest $s - t$ path.

Theorem 32. *Let $D = (V, A)$ be a digraph and $s, t \in V$. Define $D' = (V, A \cup \alpha(D)^{-1})$. Then $\mu(D) = \mu(D')$ and $\alpha(D) = \alpha(D')$.*

Proof. It suffices to show that $\mu(D)$ and $\alpha(D)$ are invariant if we add a^{-1} to D for one arc $a \in \alpha(D)$. Suppose not, then there is a directed $s - t$-path P_1 traversing a^{-1} of length at most $\mu(D)$. As $a \in \alpha(D)$ there is a path P_2 traversing a of length $\mu(D)$. If we follow P_2 until the tail of a is reached and from thereon follow P_1, we obtain a path of D of length less than $\alpha(D)$. This is a contradiction. □

of Theorem 30. Let us augment flow f along a shortest $s - t$-path P in $D(f)$ obtaining flow f'. The residual graph $D_{f'}$ is a subgraph of $D' = (V, A_f \cup \alpha(D(f))^{-1})$. Hence $\mu(D_{f'}) \geqslant \mu(D') = \mu(D(f))$. If $\mu(D_{f'}) = \mu(D(f))$, then $\alpha(D_{f'}) \subseteq \alpha(D') = \alpha(D(f))$. At least one arc of P does not belong to $D_{f'}$, (the arc of minimum residual capacity!) thus the inclusion is strict. Since $\mu(D(f))$ increases at most $|V|$ times and, as long as $\mu(D(f))$ does not change, $\alpha(D(f))$ decreases at most $2\,|A|$ times, we have the theorem. □

In the following let $m = |A|$ and $n = |V|$.

Corollary 33. *A maximum flow can be found in time $O(n\,m^2)$.*

1.2 Shortest Paths

Definition 34 (Cycle)**.** A walk in which starting node and end-node agree is called a *cycle*.

Suppose we are given a directed graph $D = (V, A)$ and a length function $c : A \longrightarrow \mathbb{R}$. The *length* of a path P is defined as

$$c(P) = \sum_{\substack{a \in A \\ a \in P}} c(a).$$

We now study how to determine a shortest path in the weighted graph D efficiently, in case of the absence of cycles of negative length.

Theorem 35. *Suppose that each cycle in D has non-negative length and suppose there exists an $s - t$-walk in D. Then there exists a path connecting s with t which has minimum length among all walks connecting s and t.*

Proof. If there exists an $s-t$-walk, then there exists an $s-t$path. Since the number of arcs in a path is at most $|A|$, there must exist a shortest *path* P connecting s and t. We claim that $c(P) \leqslant c(W)$ for all $s-t$-walks W. Suppose that there exists a an $s-t$walk W with $c(W) < c(P)$. Then let W be such a walk with a minimum number of edges. Clearly W contains a cycle C. If the cycle has nonnegative length, then it can be removed from W to obtain a walk whose length is at most $c(W)$ and whose number of edges is strictly less than $|C|$. □

We use the notation $|W|, |C|, |P|$ to denote the number of edges in a walk W a cycle C or a path P.

As a conclusion we can note here:

> If there do not exist negative cycles in D, and s and t are connected, then there exists a shortest path traversing at most $|V|-1$ arcs.

The Bellman-Ford algorithm

Let $n = |V|$. We calculate functions $f_0, f_1, \ldots, f_n : V \longrightarrow \mathbb{R} \cup \{\infty\}$ successively by the following rule.

i) $f_0(s) = 0$, $f_0(v) = \infty$ for all $v \neq s$

ii) For $k < n$ if f_k has been found, compute

$$f_{k+1}(v) = \min\{f_k(v), \min_{(u,v)\in A}\{f_k(u) + c(u,v)\}$$

for all $v \in V$.

Theorem 36. *For each $k = 0, \ldots, n$ and for each $v \in V$*

$$f_k(v) = \min\{c(P \mid P \text{ is an } s-v-\text{Path traversing at most } k \text{ arcs}.$$

Corollary 37. *If $D = (V, A)$ does not contain negative cycles w.r.t. c, then $f_n(v)$ is equal to the length of a shortest $s-v$-Path. The numbers $f_n(v)$ can be computed in time $O(|V| \cdot |A|)$.*

Corollary 38. *In time $O(|V|^2|A|)$ one can test whether $D = (V, A)$ has a negative cycle w.r.t. c and eventually return one.*

1.3 Minimum cost network flows, MCNFP

In contrast to the maximum $s-t$-flow problem, the goal here is to route a flow, which comes from several sources and sinks through a network with capacities and *costs* in such a way, that the total cost is minimized.

Example 39. *Suppose you are given a directed graph width edge weights $D = (V, A)$, $c : A \to \mathbb{R}_{\geqslant 0}$ and your task is to compute a shortest path from a particular node s to all other nodes in the graph and assume that such paths exist. Then one can model this as a MCNFP by sending a flow of value $|V|-1$ into the source node and by letting a flow of value 1 leave each node. The edges have infinite capacities.*

Here is a formal definition of a minimum cost network flow problem. In this notation, vertices are indexed with the letters i, j, k and edges (arcs) are denoted by their tail and head respectively, for example (i, j) denotes the edge from i to j.

For a node $i \in V$, the subset $I(i) \subseteq V$ denotes the incoming nodes of i, i.e., the set $I(i) = \{j \mid (j, i) \in A\} \subseteq V$. Similarly $O(i) = \{j \mid (i, j) \in A\} \subseteq V$ denotes the set of outgoing nodes. A network is now a directed graph $D = (V, A)$ together with a capacity function $u : A \to \mathbb{Q}_{\geqslant 0}$, a cost function $c : A \to \mathbb{Q}$ and an external flow $b : V \to \mathbb{Q}$. The value of $b(i)$ denotes the amount of flow which comes from the exterior. If $b(i) > 0$, then there is flow from the outside, entering the network through node i. If $b(i) < 0$, there is flow which leaves the network through i.

In the following we often use the notation $f(i, j)$ for the flow-value on the arc (i, j) (instead of $f((i, j))$). Similarly we write $c(i, j)$ and $u(i, j)$.

A *feasible flow* is a function $f : A \to \mathbb{Q}_{\geqslant 0}$ which satisfies the following constraints.

$$\begin{array}{ll} \sum_{j \in O(i)} f(i,j) - \sum_{j \in I(i)} f(j,i) = b_i & \text{for all } i \in V, \\ 0 \leqslant f(i,j) \leqslant u(i,j) & \text{for all } (i,j) \in A. \end{array}$$

The goal is to find a feasible flow with minimum cost:

$$\begin{array}{lll} \text{minimize} & \sum_{(i,j) \in A} c(i,j) f(i,j) & \\ \text{subject to} & \sum_{j \in O(i)} f(i,j) - \sum_{j \in I(i)} f(j,i) = b(i) & \text{for all } i \in V, \\ & 0 \leqslant f(i,j) \leqslant u(i,j) & \text{for all } (i,j) \in A \end{array}$$

Without loss of generality we can make the following assumptions:

1. All data (cost, supply, demand and capacity) are integral.
2. The supplies/demands at the nodes satisfy the condition $\sum_{i \in V} b(i) = 0$ and the MCNFP has a feasible solution.
3. The network contains an incapacitated directed path between every pair of nodes.
4. All arc costs are nonnegative.
5. The graph does not contain a pair of reverse arcs.

Exercise 7. Show how to transform a MCNFP on a digraph with pairs of reverse edges into a MCNFP on a digraph with no pairs of reverse edges. The number of edges and nodes should asymptotically remain the same.

Exercise 8. Prove that there exists no feasible flow unless $\sum_{i \in V} b(i) = 0$ holds.

Exercise 9. Why can we assume that the network has a path from i to j for all $i \neq j \in V$ which is incapacitated?

An *arc-flow* of D is a flow vector, that satisfies the nonnegativity and capacity constraints.

$$\sum_{j \in O(i)} f((i,j)) - \sum_{j \in I(i)} f((j,i)) = -e(i) \qquad \text{for all } i \in V,$$
$$0 \leqslant f((i,j)) \leqslant u((i,j)) \qquad \text{for all } (i,j) \in A.$$

- If $e(i) > 0$, then i is an *excess node* (more inflow than outflow).
- If $e(i) < 0$, then i is a *deficit node* (more outflow than inflow).
- If $e(i) = 0$ then i is called *balanced*.

Exercise 10. Prove that $\sum_{i \in V} e(i) = 0$.

Let $\mathscr{P}$ be the collection of directed paths of D and let $\mathcal{C}$ be the collection of directed cycles of D. A path-flow is a function $\beta : \mathscr{P} \cup \mathcal{C} \to \mathbb{R}_{\geqslant 0}$ which assigns flow values to paths and cycles.

For $(i,j) \in A$ and $P \in \mathscr{P}$ let $\delta_{(i,j)}(P)$ be 1 if $(i,j) \in P$ and 0 otherwise. For $C \in \mathcal{C}$ let $\delta_{(i,j)}(C)$ be 1 if $(i,j) \in C$ and 0 otherwise.

A path-flow β determines a unique arc-flow

$$f(i,j) = \sum_{P \in \mathscr{P}} \delta_{(i,j)}(P)\beta(P) + \sum_{C \in \mathcal{C}} \delta_{(i,j)}(C)\beta(C).$$

Theorem 40. *Every path and cycle flow has a unique representation as a nonnegative arc-flow. Conversely, every nonnegative arc flow f can be represented as a path and cycle flow with the following properties:*

1. *Every directed path with positive flow connects a deficit node with an excess node.*
2. *At most $n + m$ paths and cycles have nonzero flow and at most m cycles have nonzero flow.*

If the arc flow f is integral, then so are the path and cycle flows into which it decomposes.

Proof. "$\Rightarrow$" See discussion above.

"$\Leftarrow$"

Let f be an arc flow. Suppose i_0 is a deficit node. Then there exists an incident arc (i_0, i_1) which carries a positive flow. If i_1 is an excess node, we have found a path from deficit to excess node. Otherwise, the flow balance constraint at i_1 implies that there exists an arc (i_1, i_2) with positive flow. Repeating this procedure, we finally must arrive at an excess node or revisit a node. This means that we either have constructed a directed path P from deficit node to excess node or a directed cycle C, both involving only edges with strictly positive flow.

In the first case, let $P = i_0, \ldots, i_k$ be the directed path from deficit node i_0 to excess node i_k. We set $\beta(P) = \min\{-e_{i_0}, e_{i_k}, \min\{f(i,j) \mid (i,j) \in P\}\}$ and

$f(i,j) = f(i,j) - \beta(P)$, $(i,j) \in P$. In the second case, set $\beta(C) = \min\{f(i,j) \mid (i,j) \in C$ and $f(i,j) = f(i,j) - \beta(C)$, $(i,j) \in C$. Repeat this procedure until all node imbalances are zero.

Now find an arc with positive flow and construct a cycle C by following only positive arcs from there. Set $\beta(C) = \min\{f(i,j) \mid (i,j) \in C\}$ and $f(i,j) = f(i,j) - \beta(C)$, $(i,j) \in C\}$. Repeat this process until there are no positive flow-edges left.

Each time a path or a cycle is identified, the excess/deficit of some node is set to zero or some edge is set to zero. This implies that we decompose into at most $n + m$ paths and cycles. Since cycle detection sets an edge to zero we have at most m cycles. □

An arc flow f with $e(i) = 0$ for each $i \in V$ is called a *circulation*.

Corollary 41. *A circulation can be decomposed into at most m cycle flows.*

Let $D = (V, A)$ be a network with capacities $u(i,j)$, $(i,j) \in A$ and costs $c(i,j)$, $(i,j) \in A$ and let f be a feasible flow of the network. The *residual network* $D(f)$ is defined as follows.

- We replace each arc $(i,j) \in A$ with two arcs (i,j) and (j,i).
- The arc (i,j) has cost $c(i,j)$ and *residual capacity* $r(i,j) = u(i,j) - f(i,j)$.
- The arc (j,i) has cost $-c(i,j)$ and residual capacity $r(j,i) = f(i,j)$.
- Delete all arcs which do not have strictly positive residual capacity.

A directed cycle in $D(f)$ is called an *augmenting cycle* of f.

Exercise 11. Suppose that f and f° are feasible flows. Prove that $f - f^\circ$ is a circulation in $D(f^\circ)$. Here $f - f^\circ$ is the flow

$$(f - f^\circ)(i,j) = \begin{cases} \max\{0, f(i,j) - f^\circ(i,j)\}, & \text{if } (i,j) \in A(D) \\ \max\{0, f^\circ(j,i) - f(j,i)\}, & \text{if } (j,i) \in A(D) \\ 0, & \text{otherwise.} \end{cases}$$

Theorem 42 (Augmenting Cycle Theorem)**.** *Let f and f° be any two feasible flows of a network flow problem. Then f equals f° plus the flow of at most m directed cycles in $D(f^\circ)$. Furthermore the cost of f equals the cost of f° plus the cost of flow on these augmenting cycles.*

Proof. This can be seen by applying flow decomposition on the flow $f - f^\circ$ in $D(f^\circ)$. □

Theorem 43 (Negative Cycle Optimality Conditions)**.** *A feasible flow f^* is an optimal solution of the minimum cost network flow problem, if and only if it satisfies the negative cycle optimality conditions: The residual network $D(f^*)$ contains no directed cycle of negative cost.*

Proof. "$\Rightarrow$" Suppose that f is a feasible flow and that $D(f)$ contains a negative directed cycle. Then f cannot be optimal, since we can augment positive flow along the corresponding cycle in the network. Therefore, if f^* is an optimal flow, then $D(f^*)$ cannot contain a negative directed cycle.

"$\Leftarrow$" Suppose now that f^* is a feasible flow and suppose that $D(f^*)$ does not contain a negative cycle. Let f° be an optimal flow with $f^\circ \neq f^*$. The vector $f^\circ - f^*$ is a circulation with nonpositive cost $c^T(f^\circ - f^*) \leqslant 0$. It follows from Theorem 42 that the cost of f° equals the cost of f^* plus the cost of directed cycles in the residual network $D(f^*)$. The cost of these cycles is nonnegative, and therefore $c(f^\circ) \geqslant c(f^*)$ which implies that f^* is optimal.

□

Exercise 12. What is the maximum amount of flow, which can be augmented along a cycle in the residual network. Describe the flow after the augmentation and prove that it is feasible.

We now analyze the MMCC-algorithm. Let $\mu(f)$ denote the minimum mean-weight of a cycle in $D(f)$.

Lemma 44 (See Korte & Vygen [3]). *Let $f_1, f_2, \ldots$ be a sequence of feasible flows such that f_{i+1} results from f_i by augmenting flow along C_i, where C_i is a minimum mean cycle of $D(f_i)$, then*

1. *$\mu(f_k) \leqslant \mu(f_{k+1})$ for all k.*
2. *$\mu(f_k) \leqslant \frac{n}{n-1}\mu(f_l)$, where $k < l$ and $C_k \cup C_l$ contains a pair of reversed edges.*

Proof. 1): Suppose f_k and f_{k+1} are two subsequent flows in this sequence. Consider the multi-graph H which results from C_k and C_{k+1} by deleting pairs of opposing edges. The arcs of H are a subset of the arcs of $D(f_k)$, since an arc of C_{k+1} which is not in $D(f_k)$ must be a reverse arc of C_k.

Each node in H has even degree. Thus H can be decomposed into cycles, each of mean weight at least $\mu(f_k)$. Thus we have $c(A(H)) \geqslant \mu(f_k)|A(H)|$.

Since the total weight of each reverse pair of edges is zero we have

$$c(A(H)) = c(C_k) + c(C_{k+1}) = \mu(f_k)|C_k| + \mu(f_{k+1})|C_{k+1}|.$$

Since $|A(H)| \leqslant |C_k| + |C_{k+1}|$ we conclude

$$\begin{aligned} \mu(f_k)(|C_k| + |C_{k+1}|) &\leqslant \mu(f_k)|A(H)| \\ &\leqslant c(A(H)) \\ &= \mu(f_k)|C_k| + \mu(f_{k+1})|C_{k+1}|. \end{aligned}$$

Thus $\mu(f_k) \leqslant \mu(f_{k+1})$.

2): By the first part of the theorem, it is enough to prove the statement for k, l such that $C_i \cup C_l$ does not contain a pair of reverse edges for each i, $k < i < l$.

Again, consider the graph H resulting from C_k and C_l by deleting pairs of opposing edges. H is a subgraph of $D(f_k)$, since any edge of C_l which does not belong to $D(f_k)$ must be a reverse edge of $C_k, C_{k+1}, \ldots, C_{l-1}$. But only C_k contains a reverse edge of C_l. So as above we have

$$c(A(H)) = c(C_k) + c(C_l) = \mu(f_k)|C_k| + \mu(f_l)|C_{k+1}|.$$

Since $|A(H)| \leqslant |C_k| + |C_l| - 2$ we have $|A(H)| \leqslant \frac{n-1}{n}(|C_k| + |C_l|)$. Thus we get

$$\begin{aligned} \mu(f_k)\frac{n-1}{n}(|C_k| + |C_l|) &\leqslant \mu(f_k)|A(H)| \\ &\leqslant c(A(H)) \\ &= \mu(f_k)|C_k| + \mu(f_l)|C_l| \\ &\leqslant \mu(f_l)(|C_k| + |C_l|) \end{aligned}$$

This implies that $\mu(f_k) \leqslant \frac{n}{n-1}\mu(f_l)$.

□

Corollary 45. *During the execution of the MMCC-algorithm, $|\mu(f)|$ decreases by a factor of $1/2$ every $n \cdot m$ iterations.*

Proof. Let $C_1, C_2, \ldots$ be the sequence of augmenting cycles. Every m-th iteration, there must be an edge of the cycle, which is reverse to one of the succeeding $m-1$ cycles, because every iteration, one edge of the residual network will be deleted. Thus after $n\,m$ iterations, the absolute value of μ has dropped by $\left(\frac{n-1}{n}\right)^n \leqslant e^{-1} \leqslant 1/2$. □

Corollary 46. *If all data are integral, then the MMCC-algorithm runs in polynomial time.*

Proof.

- A lower bound on μ is the smallest cost c_{min}
- $|\mu|$ drops by $1/2$ every $m\,n$ iterations.
- After $mn \log n |c_{min}|$ iterations, absolute value of minimum mean weight cycle drops below $1/n$, thus is zero.
- **We need to prove that a minimum mean cycle can be found in polynomial time**

□

This is a so-called *weakly polynomial* bound, since the binary encoding length of the numbers in the input (here the costs) influences the running time. We now prove that the MMCC-algorithm is *strongly polynomial*.

Theorem 47 (See Korte & Vygen [3])**.** *The MMCC-algorithm requires $O(m^2\, n \log n)$ iterations (mean weight cycle cancellations).*

Proof. One shows that every $m\,n(\lceil \log n \rceil + 1)$ iterations, at least one edge is *fixed*, which means that the flow through this edge does not change anymore.

Let f_1 be some flow at some iteration and let f_2 be the flow $m\,n(\lceil \log n \rceil + 1)$ iterations later. It follows from Corollary 45 that

$$\mu(f_1) \leqslant 2\,n\,\mu(f_2) \tag{1.21}$$

holds.

Define the costs $c'(e) = c(e) - \mu(f_2)$ for the residual network $D(f_2)$. There exists no negative cycle in $D(f_2)$ w.r.t. this cost c'. (A cycle C has weight $c'(C) = \sum_{e \in C} c(e) - |C|\mu(f_2)$ and thus $c'(C)/|C| = \sum_{e \in C} c(e)/|C| - \mu(f_2) \geqslant 0$). By Lemma **??** there exists a feasible node potential π for these weights. One has $0 \leqslant c'_\pi(e) = c_\pi(e) - \mu(f_2)$ and thus

$$c_\pi(e) \geqslant \mu(f_2), \text{ for all } e \in A(D(f_2)). \tag{1.22}$$

Let C be a minimum mean cycle of $D(f_1)$. One has

$$c_\pi(C) = c(C) = \mu(f_1)\,|C| \leqslant 2\,n\,\mu(f_2)|C|. \tag{1.23}$$

It follows that there exists an arc e_0 of C such that

$$c_\pi(e_0) \leqslant 2\,n\,\mu(f_2) \tag{1.24}$$

holds. The inequalities (1.22) imply that $e_0 \notin A(D(f_2))$

We now make the following claim:

Let f' be a feasible flow such that $e_0 \in D(f')$, then $\mu(f') \leqslant \mu(f_2)$.

If we have shown this claim, then it follows from Lemma 44 that e_0 cannot be anymore in the residual network of a flow after f_2. Thus the flow along the edge e_0 (or e_0^{-1}) is fixed.

Let f' be a flow such that $e_0 \in A(D(f'))$. Recall that $f' - f_2$ is a circulation in $D(f_2)$ where $e_0 \notin D(f_2)$, $e_0^{-1} \in D(f_2)$ and this circulation sends flow over e_0^{-1}. This circulation can be decomposed into cycles and one of these cycles C contains e_0^{-1}. One has $c_\pi(e_0^{-1}) = -c_\pi(e_0) \geqslant -2\,n\,\mu(f_2)$ (eq. (1.24)). Using (1.22) one obtains

$$\begin{aligned} c(C) &= \sum_{e \in C} c_\pi(e) && (1.25)\\ &\geqslant -2\,n\,\mu(f_2) + (n-1)\mu(f_2) && (1.26)\\ &= -(n+1)\,\mu(f_2) && (1.27)\\ &> -n\,\mu(f_2). && (1.28) \end{aligned}$$

The reverse of C is an augmenting cycle for f' with total weight at most $n\,\mu(f_2)$ and thus with mean weight at most $\mu(f_2)$. Thus $\mu(f') \leqslant \mu(f_2)$. □

1.4 Computing a minimum cost-to-profit ratio cycle

Given a digraph $D = (V, A)$ with costs $c : A \to \mathbb{Z}$ and profit $p : A \to \mathbb{N}_{>0}$, the task is to compute a cycle $C \in \mathcal{C}$ with minimum ratio

$$\frac{c(C)}{p(C)}. \tag{1.29}$$

Notice that this is the largest number $\beta \in \mathbb{Q}$ which satisfies

$$\beta \leqslant \frac{c(C)}{p(C)}, \text{ for all } C \in \mathcal{C}. \tag{1.30}$$

By rewriting this inequality, we understand this to be the largest number $\beta \in \mathbb{Q}$ such that

$$c(C) - \beta\, p(C) \geqslant 0 \text{ for all } C \in \mathcal{C}. \tag{1.31}$$

In other words, we search the largest number $\beta \in \mathbb{Q}$ such that the digraph $D = (V, A)$ with costs $c_\beta : A \to \mathbb{Q}$, where $c_\beta(e) = c(e) - \beta\, p(e)$.

We need a routine to check whether D has a negative cycle for a given weight function c. For this we assume w.l.o.g. that each vertex is reachible from the vertex s, if necessary by intruducing a new vertex s from which there is an edge with cost and profit 0 to all other nodes. The minimum cost-to-profit ration cycle w.r.t. this new graph is then the minimum cost to profit ratio cycle w.r.t. the original graph, since s is not a vertex of any cycle.

Recall the following single-source shortest-path algorithm of Bellman-Ford which we now apply with weights c_β:

> Let $n = |V|$. We calculate functions $f_0, f_1, \ldots, f_n : V \longrightarrow \mathbb{R} \cup \{\infty\}$ successively by the following rule.
>
> i) $f_0(s) = 0$, $f_0(v) = \infty$ for all $v \neq s$
>
> ii) For $k < n$ if f_k has been found, compute
>
> $$f_{k+1}(v) = \min\{f_k(v), \min_{(u,v)\in A}\{f_k(u) + c_\beta(u, v)\}$$
>
> for all $v \in V$.

There exists a negative cycle w.r.t. c_β if and only if $f_n(v) < f_k(v)$ for some $v \in V$ and $1 \leqslant k < n$. Thus we can test in $O(m \cdot n)$ steps whether D, c_β contains a negative cycle.

We now apply the following idea to search for the correct value of β. We keep an interval $I = [L, U]$ with the invariant that the value β that we are searching lies in this interval I. As starting values, we can chose $L = c_{min}$ and $U = c_{max}$, where c_{min} and c_{max} are the smallest and largest cost respectively. In one iteration we compute $M = (L + U)/2$. We then check whether D, together with c_M contains a negative cycle. If yes, we know that β is at least M and we set $L \leftarrow M$. If not, then β is at most M and we update the upper bound $U \leftarrow M$.

When can we stop this procedure? We can stop it, if we can assure that only one valid cost-to-profit ratio cycle lies in $[L, U]$. Suppose that C_1 and C_2 have different cost-to-profit ratios. Then

$$|c(C_1)/p(C_1) - c(C_2)/p(C_2)| = \left|\frac{c(C_1)\,p(C_2) - c(C_2)p(C_1)}{(p(C_1)\,p(C_2))}\right| \quad (1.32)$$

$$\geqslant 1/(n^2 p_{max}^2). \quad (1.33)$$

Thus we can stop our process, if $U - L < 1/(n^2 p_{max}^2)$, since we know then that there can be only one cycle $c \in \mathcal{C}$ with $c(C)/p(C) \in [L, U]$.

Suppose that $[L, U]$ is the final interval. We know then that

$$L \leqslant c(C)/p(C) \text{ for all } C \in \mathcal{C}$$

and

$$U > c(C)/p(C) \text{ holds for some } C \in \mathcal{C}.$$

Let C be a minimum weight cycle w.r.t. the edge costs c_L. Clearly $U > c(C)/p(C) \geqslant L$ holds and thus C is the minimum cost-to-profit cycle we have been looking for.

Let us analyze the number of required iterations. We need to halve the starting interval-length $2\,c$, where c is the largest absolute value of a cost, until the length is at most $1/(n^2 p_{max}^2)$. We search the minimal $i \in \mathbb{N}$ such that

$$(1/2)^i c \leqslant 1/(n^2 p_{max}^2). \quad (1.34)$$

This shows us that we need $O(\log(c\,p_{max}^2 n^2))$ iterations which is $O(\log n \log K)$, where K is the largest absolute value of a cost or a profit.

Theorem 48 (Lawler [4]). *Let D be a digraph with costs $c : A \to \mathbb{Z}$ and profit $p : A \to \mathbb{N}_{>0}$ an let $K \in \mathbb{N}$ such that $|c(e)| + |p(e)| \leqslant K$ for all $e \in \mathbb{N}$. A minimum cost-to-profit ratio cycle of G can be computed in time $O(m\,n\,\log n\,\log K)$.*

But we knew a weakly polynomial algorithm for MCNFP from the exercises. So you surely ask: Can we do better for minimum cost-to-profit cycle computation? The answer is "Yes"!

1.4.1 Parametric search

Let us first roughly describe the idea on how to obtain a strongly polynomial algorithm, see [6]. The Bellman-Ford algorithm tells us whether our current β is too large or too small, depending on whether D with weights c_β contains a negative cycle or not. Recall that the B-F algorithm computes labels $f_i(v)$ for $v \in V$ and $1 \leqslant i \leqslant n$. If these labels are computed with costs c_β, then they are *piecewise linear* functions in β and we denote them by $f_i(v)[\beta]$.

Denote the optimal β that we look for by β^* and suppose that we know an interval I with such that $\beta^* \in I$ and each function $f_i(v)[\beta]$ is linear if it is restricted to this domain I. Then we can determine β^* as follows.

Let $I = [L, U]$ be the interval and remember that we are searching for the largest value of $\beta \in I$ such that $f_n(v)[\beta] = f_{n-1}(v)[\beta]$ holds for each $v \in V$.

Clearly this holds for $\beta = L$. Thus we only need to check whether $\beta = U$ by computing the values $f_n(v)[U]$ and $f_{n-1}(v)[U]$ for each $v \in V$ and check whether one of these pairs consists of different numbers.

The idea is now to compute such an interval $I = [L, U]$ in strongly polynomial time.

Consider the function $f_1(v)[\beta]$. Clearly one has

$$f_1(v)[\beta] = \begin{cases} c(s,v) - \beta \cdot p(s,v) & \text{if } (s,v) \in A, \\ \infty & \text{otherwise.} \end{cases}$$

This shows that $f_1(v)[\beta]$ is a linear function in β for each $v \in V$.

Now suppose that $i \geqslant 1$ and that we have computed an interval $I = [L, U]$ with $\beta^* \in I$ and each function $f_i(v)[\beta]$ is a linear function if β is restricted to I.

Now consider the function $f_{i+1}(v)[\beta]$ for a particular $v \in V$. Recall the formula

$$f_{i+1}(v)[\beta] = \min\{f_i(v)[\beta], \min_{(u,v)\in A}\{f_i(u)[\beta] + c(u,v) - \beta \cdot p(u,v)\}\}. \tag{1.35}$$

Each of the functions $f_i(v)[\beta]$ and $f_i(u)[\beta] + c(u,v) - \beta \cdot p(u,v)$ are linear on I. The function $f_i(v)[\beta]$ can be retrieved by computing a shortest path $P_i(v)$ from s to v with edge weights c_β for some β in (L, U) which uses at most i arcs. If β is then allowed to vary, the line which is defined by $f_i(v)[\beta]$ on I is then the length of this path P with parameter β. Similarly we can retrieve the functions (lines) $f_i(u)[\beta] + c(u,v) - \beta \cdot p(u,v)$ for each $(u,v) \in A$. With the Bellman-Ford algorithm, this amounts to a running time of $O(m \cdot n)$.

We now have n lines and can now compute the lower envelope of these lines in time $O(n \log n)$ alternatively we can also compute all intersection points of these lines and sort them w.r.t. increasing β-coordinate. This would amount to $O(n^2 \log n)$. Let $\beta_1, \ldots, \beta_k$ be the sorted list of these β-coordinates. Now $\beta_{trial} := \beta_{\lfloor k/2 \rfloor}$ and check whether $\beta^* > \beta_{trial}$. If yes, we can replace L by β_{trial} and we can delete the numbers $\beta_1, \ldots, \beta_{\lfloor k/2 \rfloor - 1}$. Otherwise, we replace U by β_{trial} and delete $\beta_{\lfloor k/2 \rfloor + 1}, \ldots, b_k$. In any case, we halved the number of possible β-coordinates and continue in this way. Such a check requires a negative cycle test in the graph D with edge weights β_{trial} and costs $O(m \cdot n)$. In the end we have two consecutive β-coordinates and have an interval $[L, U]$ on which $f_{i+1}(v)[\beta]$ is linear. To find an interval I such that $f_{i+1}(v)[\beta]$ is linear on I and $\beta^* \in I$ costs thus $O(m \cdot n \log n)$ steps.

We now continue to tighten *this interval* such that all functions $f_{i+1}(v)[\beta], v \in V$ are linear on $[L, U]$. Thus in step $i + 1$ this amounts to a running time of

$$O\left(n \cdot (m \cdot n \log n)\right).$$

The total running time is thus

$$O(n^3 \cdot m \cdot \log n).$$

Theorem 49. *Let $D = (V, A)$ be a directed graph and let $c : A \longrightarrow \mathbb{R}$ and $p : A \longrightarrow \mathbb{R}_{>0}$ be functions. One can compute a cycle C of D minimizing $c(C)/p(C)$ in time $O(n^3 \cdot m \cdot \log n)$.*

Bibliography

[1] J. Edmonds. Maximum matching and a polyhedron with 0,1-vertices. *Journal of Research of the National Bureau of Standards*, 69:125–130, 1965.

[2] J. Edmonds. Paths, trees and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[3] B. Korte and J. Vygen. *Combinatorial optimization*, volume 21 of *Algorithms and Combinatorics*. Springer-Verlag, Berlin, second edition, 2002. Theory and algorithms.

[4] E. L. Lawler. *Combinatorial optimization: networks and matroids*. Holt, Rinehart and Winston, New York, 1976.

[5] L. Lovász. Graph theory and integer programming. *Annals of Discrete Mathematics*, 4:141–158, 1979.

[6] N. Megiddo. Combinatorial optimization with rational objective functions. *Math. Oper. Res.*, 4(4):414–424, 1979.

[7] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley, 1986.

[8] A. Schrijver. *Combinatorial optimization. Polyhedra and efficiency (3 volumes)*. Algorithms and Combinatorics 24. Berlin: Springer., 2003.