Chapter 6
Paths, cycles and flows in graphs

Suppose you want to find a shortest path from a given starting point to a given destination. This is a common scenario in driver assistance systems (GPS) and can be modeled as one of the most basic combinatorial optimization problems, the *shortest path problem*. In this chapter, we introduce directed graphs, shortest paths and flows in networks. We focus in particular on the maximum-flow problem, which is a linear program that we solve with direct methods, versus the simplex method, and analyze the running time of these direct methods.

6.1 Growth of functions

In the analysis of algorithms, it is more appropriate to investigate the asymptotic running time of an algorithm depending on the input and not the precise running time itself. We review the O, Ω and Θ-notation.

Definition 6.1 **(O, Ω, Θ-notation).**
Let $T, f : \mathbb{N} \to \mathbb{N}$ be two functions

- $T(n)$ is in $O(f(n))$, if there exist positive constants $n_o \in \mathbb{N}$ and $c \in \mathbb{R}_{>0}$ with $T(n) \leqslant c \cdot f(n)$ for all $n \geqslant n_0$.
- $T(n)$ is in $\Omega(f(n))$, if there exist constants $n_o \in \mathbb{N}$ and $c \in \mathbb{R}_{>0}$ with $T(n) \geqslant c \cdot f(n)$ for all $n \geqslant n_0$.
- $T(n)$ is in $\Theta(f(n))$ if $T(n)$ is both in $O(f(n))$ and in $\Omega(f(n))$.

Example 6.1. The function $T(n) = 2n^2 + 3n + 1$ is in $O(n^2)$, since for all $x \geqslant 1$ one has $2n^2 + 3n + 1 \leqslant 6n^2$. Here $n_0 = 1$ and $c = 6$.

Similarly $T(n) = \Omega(n^2)$, since for each $n \geqslant 1$ one has $2n^2 + 3n + 1 \geqslant n^2$. Thus $T(n)$ is in $\Theta(n^2)$.

6.2 Graphs

Definition 6.2. A *directed graph* is a tuple $G = (V, A)$, where V is a finite set, called the *vertices* of G and $A \subseteq (V \times V)$ is the set of *arcs* of G. We denote an arc by its two defining nodes $(u, v) \in A$. The nodes u and v are called *tail* and *head* of the arc (u, v) respectively.

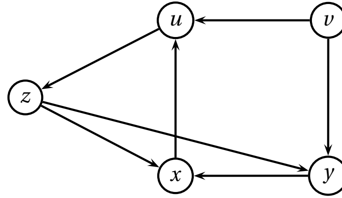


Fig. 6.1: Example of a directed graph with 5 nodes and 7 arcs.

Definition 6.3 (Walk, path, distance). A *walk* is a sequence of the form

$$P = (v_0, a_1, v_1, \dots, v_{m-1}, a_m, v_m),$$

where $a_i = (v_{i-1}, v_i) \in A$ for $i = 1, \dots, m$. If the nodes $v_0, \dots, v_m$ 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. It is denoted by $d(u, v)$.

Example 6.2. The following is a walk and a path of the graph in Figure 6.1.

$$u, (u,z), z, (z,x), x, (x,u), u, (u,z), z, (z,y), y$$
$$u, (u,z), z, (z,y), y$$

6.3 Representing graphs and computing the distance of two nodes

We represent a graph with n vertices as an array $A[v_1, \dots, v_n]$, where the entry $A[v_i]$ is a pointer to a linked list of vertices, the *neighbors of* v_i. $N(v_i) = \{u \in V : (v_i, u) \in A\}$.

We next describe a very basic algorithm, which computes the distances from $s \in V$ to all other nodes. We let $V_i \subseteq V$ be the set of vertices which have distance i from s.

Lemma 6.1. *For $i = 0, \dots, n-1$, the set V_{i+1} is equal to the set of vertices $v \in V \setminus (V_0 \cup \dots \cup V_i)$ such that there exists an arc $(u, v) \in A$ with $u \in V_i$.*

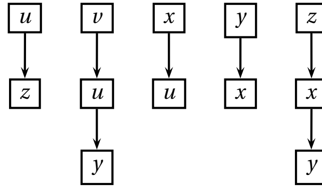


Fig. 6.2: Adjacency list representation of the graph in Figure 6.1.

Proof. Suppose that $v \notin V_0 \cup \cdots \cup V_i$ and there exists an arc $uv \in A$ with $u \in V_i$. Since $u \in V_i$, there exists a path $s, a_1, v_1, a_2, v_2, \ldots, a_i, u$ of length i from s to u. The sequence $s, a_1, v_1, a_2, v_2, \ldots, a_i, u, uv, v$ is a path of length $i+1$ from s to v and thus $v \in V_{i+1}$.

If, on the other hand, $v \in V_{i+1}$, then there exists a path

$$s, a_1, v_1, \ldots, a_i, v_i, a_{i+1}, v$$

of length $i+1$ from s to v. We need to show that $v_i \in V_i$ holds. Clearly, since there exists a path of length i from s to v_i, one has $v_i \in V_j$ with $j \leqslant i$. If $j < i$, then there exists a path $s, a'_1, v'_1, \ldots, a'_j, v_i$ of length j which can be extended to a path of length $j+1 < i+1$ from s to v

$$s, a'_1, v'_1, \ldots, a'_j, v_i, a_{i+1}, v$$

which contradicts $v \in V_{i+1}$. □

We now describe the *breadth-first search algorithm*, that computes the distances from a starting vertex s to all other vertices $v \in V$.

The algorithm maintains arrays

$$D[v_1 = s, v_2, \ldots, v_n]$$
$$\pi[v_1 = s, v_2, \ldots, v_n]$$

and a queue Q which contains only s in the beginning. The array D contains at termination of the algorithm the distances from s to all other nodes and is initialized with $[0, \infty, \ldots, \infty]$. The array π contains predecessor information for shortest paths, in other words, when the algorithm terminates, $\pi[v] = u$, where uv is an arc and $D[u] + 1 = D[v]$. The array π is initialized with $[0, \ldots, 0]$.

After this initialization, the algorithm proceeds as follows.

while $Q \neq \emptyset$
    $u := head(Q)$
    for each $v \in N(u)$
        if $(D[v] = \infty)$
            $\pi[v] := u$

$D[v] := D[u]+1$
$enqueue(Q, v)$
$dequeue(Q)$

Here the function $head(Q)$ returns the next element in the queue and $dequeue(Q)$ removes the first element of Q, while $enqueue(Q, v)$ adds v to the queue as last element.

Lemma 6.2. *The breadth-first search algorithm assigns* distance labels *D correctly.*

Proof. Let $v \in V$. We show by induction on $d(s, v)$ that the labels are correctly assigned.

If $d(s, v) = 0$, then $s = v$ and $D[v] = 0$. If $d(s, v) = 1$, then v is a neighbor of s and $D[v] = 1$ is set correctly in the first iteration of the **while** loop.

Let $d(s, v) > 1$. Then there exists a $u \neq s, v$ with $d(s, u) = d(s, v) - 1$ and $uv \in A$. By induction, the label $D[u] = d(s, u)$ is set correctly by the breadth-first-search algorithm. Also, since the breadth-first-search algorithm computes for v a path of length $D[v]$ from s to v, the node v receives a label which is greater than or equal to $d(s, v)$. If we consider the sequence (over time) of assigned labels, that breadth-first-search is assigning, then it is easy to see that this sequence is monotonously increasing, see exercise 6. The node v is thus explored at the latest, when u is dequeued. This shows that the label of v, $D[v]$ is assigned correctly. □

Definition 6.4 (Tree). A *directed tree* is a directed graph $T = (V, A)$ with $|A| = |V| - 1$ and there exists a node $r \in T$ such that there exists a path from r to all other nodes of T.

Lemma 6.3. *Consider the arrays D and π after the termination of the breadth-first-search algorithm. The graph $T = (V', A')$ with $V' = \{v \in V : D[v] < \infty\}$ and $A' = \{\pi(v)v : 1 \leqslant D[v] < \infty\}$ is a tree.*

Definition 6.5. The tree T from above is the *shortest-path-tree* of the (unweighted) directed graph $G = (V, A)$.

Theorem 6.1. *The breath-first-search algorithm runs in time $O(|V| + |A|)$.*

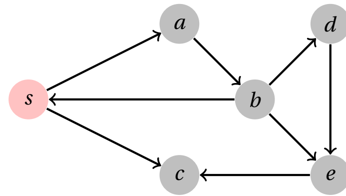
Proof. Each vertex is queued and dequeued at most once. These queuing operations take constant time each. Thus queuing and dequeuing costs $O(|V|)$ in total.

When a vertex u is dequeued, its neighbors are inspected and the operations in the **if** statement cost constant time each. Thus one has an additional cost of $O(|A|)$, since these constant-time operations are carried out for each arc $a \in A$.

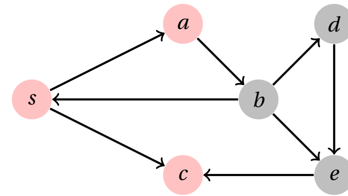
□

6.4 Shortest Paths

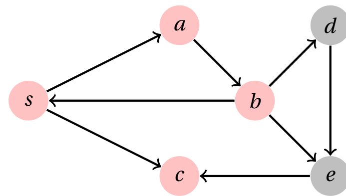
Definition 6.6 (Cycle). A walk in which starting node and end-node agree is called a *cycle*.



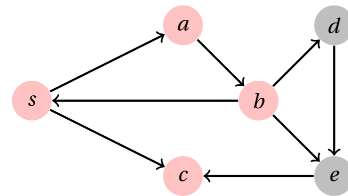
(a) The breadth-first search algorithm starts with the queue $Q = [s]$. The distance labels for $[s, a, b, c, d, e]$ are $[0, \infty, \infty, \infty, \infty, \infty]$ respectively.



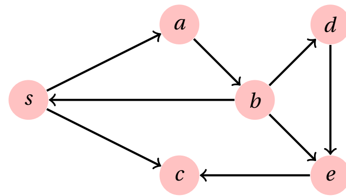
(b) After the first iteration of the **while** loop the queue is $Q = [a, c]$ and the distance labels are $[0, 1, \infty, 1, \infty, \infty]$ respectively.



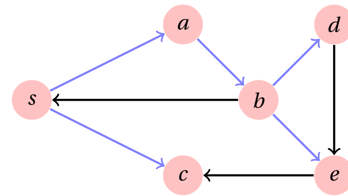
(c) After the second iteration of the **while** loop the queue is $Q = [c, b]$ and the distance labels are $[0, 1, 2, 1, \infty, \infty]$ respectively.



(d) After the third iteration of the **while** loop the queue is $Q = [b]$ and the distance labels are unchanged, since c does not have any neighbors.



(e) After the fourth iteration of the **while** loop the queue is $Q = [d, e]$ and the distance labels are $[0, 1, 2, 1, 3, 3]$ respectively.



(f) After the sixth iteration of the **while** loop the queue is empty $Q = []$ and the distance labels remain unchanged. The blue edges denote the shortest path tree.

Fig. 6.3: An example-run of breadth-first search

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

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

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 6.2. *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 an $s-t$-walk W with $c(W) < c(P)$. Then let W be such a walk with a minimum number of arcs. 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 arcs is strictly less than $|C|$. □

We use the notation $|W|, |C|, |P|$ to denote the number of arcs 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 walk 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

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

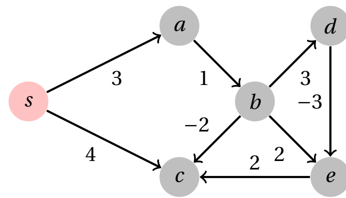
for all $v \in V$.

Theorem 6.3. *For each $k = 0, \ldots, n$ and for each $v \in V$*

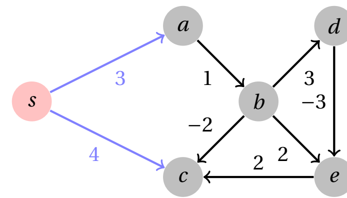
$$f_k(v) = \min\{c(P) : P \text{ is an } s-v\text{-walk traversing at most } k \text{ arcs}\}.$$

Corollary 6.1. *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 $O(|V| \cdot |A|)$.*

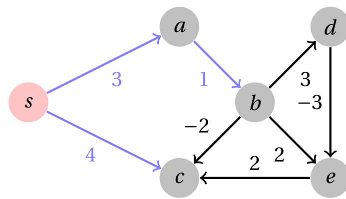
Corollary 6.2. *In time $O(|V|^2|A|)$ one can test whether $D = (V, A)$ has a negative cycle w.r.t. c and eventually return one.*



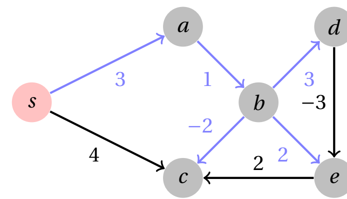
(a) The algorithm is initialized with distance labels for s, a, b, c, d, e being $[0,\infty,\infty,\infty,\infty,\infty]$ respectively



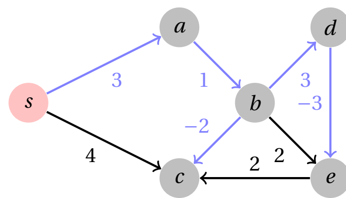
(b) After the first iteration the labels are $[0,3,\infty,4,\infty,\infty]$



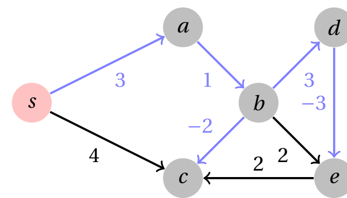
(c) After the second iteration the labels are $[0,3,4,4,\infty,\infty]$



(d) After the third iteration the labels are $[0,3,4,2,7,6]$



(e) After the fourth iteration the labels are $[0,3,4,2,7,4]$



(f) After the fifth iteration the labels are unchanged. The shortest path distances have been computed.

Fig. 6.4: An example-run of the Bellman-Ford algorithm. The blue edges represent the tree whose paths have the corresponding lengths.

6.5 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 6.7 (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

$$\sum_{e\in\delta^{out}(v)} f(e) = \sum_{e\in\delta^{in}(v)} f(e), \text{ for all } v \in V - \{s,t\}, \tag{6.1}$$

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. Arc sets of the form $\delta^{out}(U)$ are called a *cut* of D. The *capacity of a cut* $u(\delta^{out}(U))$ is the sum of the capacities of its arcs.

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

$$\max \sum_{e\in\delta^{out}(s)} x(e) - \sum_{e\in\delta^{in}(s)} x(e) \tag{6.2}$$

$$\sum_{e\in\delta^{out}(v)} x(e) = \sum_{e\in\delta^{in}(v)} x(e), \text{ for all } v \in V-\{s,t\} \tag{6.3}$$

$$x(e) \leqslant u(e), \text{ for all } e \in A \tag{6.4}$$

$$x(e) \geqslant 0, \text{ for all } e \in A \tag{6.5}$$

Definition 6.8 (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 6.4. *Let $D = (V, A)$ be a digraph, let $f : A \to \mathbb{R}$ and let $U \subseteq V$, then*

$$excess_f(U) = \sum_{v\in U} excess_f(v). \tag{6.6}$$

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 its tail in U is subtracted once on the right and once on the left. An arc which has its 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 6.5 (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 6.9 (Residual graph). Let $f : A \to \mathbb{R}$, and $u : A \to \mathbb{R}$ where $0 \leqslant f \leqslant u$. Consider the sets of arcs

$$A_f = \{a \mid a \in A, f(a) < u(a)\} \cup \{a^{-1} \mid a \in A, f(a) > 0\}. \tag{6.7}$$

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

Corollary 6.3. *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 $\delta^{out}(U)$ 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 does not carry any flow and thus $f(\delta^{in}(U) = 0$. It follows that $value(f) = u(\delta^{out}(U))$ and f is optimal by Theorem 6.5. □

Definition 6.10 (undirected walk). An *undirected walk* is a sequence of the form $P = (v_0, a_1, v_1, \dots, v_{m-1}, a_m, v_m)$, where $a_i \in A$ for $i = 1, \dots, m$ and $a_i = (v_{i-1}, v_i)$ or $a_i = (v_i, v_{i-1})$. If the nodes $v_0, \dots, v_m$ are all different, then P is an *undirected path.*

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

$$\chi^P(a) = \begin{cases} 1 & \text{if } P \text{ traverses } a, \\ -1 & \text{if } P \text{ traverses } a^{-1}, \\ 0 & \text{if } P \text{ traverses neither } a \text{ or } a^{-1}. \end{cases} \tag{6.8}$$

Theorem 6.6 (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 $D(f)$. 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 $D(f)$.

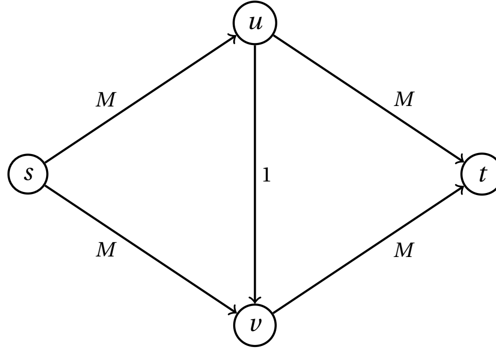
Let U be the nodes reachable from s in $D(f)$. Then $value(f) = u(\delta^{out}(U))$ and $\delta^{out}(U)$ is an $s-t$-cut of minimum capacity by the weak duality theorem.

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

Theorem 6.7. *If all capacities are rational, this algorithm terminates.*



The example above shows that, if the augmenting paths are chosen in a disadvantageous way, then the Ford-Fulkerson algorithm may take $\Omega(M)$ iterations, where M is the largest capacity in the network. This happens if all augmenting paths use the arc uv or vu respectively in the residual network.

Corollary 6.4 (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 $a \in A$).*

Proof. This follows from the fact that the residual capacities remain integral and thus the augmented flow is always integral. □

Theorem 6.8. *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 6.11. 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 6.9. *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 another $s-t$ path P_3 in D. Similarly if we follow P_1 until the head of a is reached and then follow P_2, we obtain a fourth $s-t$ path P_4 in D. However P_3 or P_4 has length less than $\mu(D)$. This is a contradiction. □

Proof (of Theorem 6.8). 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 6.5. *A maximum flow can be found in time $O(n\,m^2)$.*

6.6 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 6.3. Suppose you are given a directed graph width arc 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 costs on the arcs are defined by c. The arcs have infinite capacities. We will see later, that this minimum cost network flow problem has an integral solution which corresponds to the shortest paths from s to all other nodes.

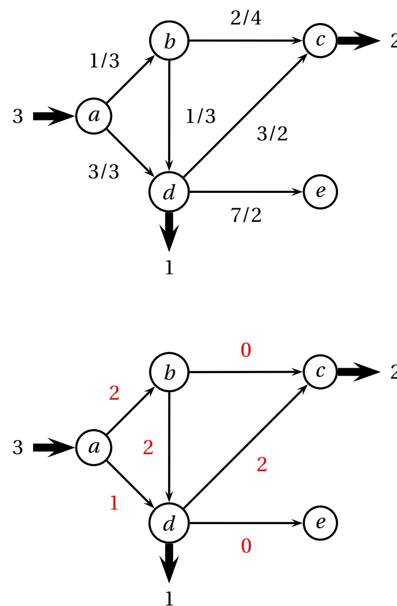


Fig. 6.5: A Network with in/out-flow, costs and capacities and a feasible flow of cost 13.

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 arcs are denoted by their tail and head respectively, for example (i, j) denotes the arc from i to j.

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.

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

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

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

Example 6.4. Imagine you are a pilot and fly a passenger airplane in hops from airport 1 to airport 2 to airport 3 and so on, until airport n. At airport i there are b_{ij} passengers that want to travel to airport j, where $j > i$. You may decide how many of the b_{ij} passengers you will take on board. Each of the passengers will pay c_{ij} dollars for the trip. The airplane can accommodate p people.

You are a greedy pilot and think of a plan to pick up and deliver passengers on your hop from 1 to n which maximizes your revenue.

Finding this plan can be modeled as a minimum cost network flow problem. Your network has nodes $1, \ldots, n$ and arcs $(i, i+1), i = 1, \ldots, n-1$ with capacities p and without costs. These nodes do not have in/out-flow from the outside. You furthermore have nodes $i \to j$ for $i < j$ and $i, j \in \{1, \ldots, n\}$ which are excess nodes with in-flow b_{ij} from the outside. Each node $i \to j$ is connected to i and to j with a directed arc. The capacities on these arcs are infinite. The cost of the arc $(i \to j, i)$ is $-c_{ij}$. The cost of the arc $(i \to j, j)$ is zero. The outflow on the node j is the total number of passengers that want to fly to node j. An integral optimal flow to this problem is an optimal plan for you.

Throughout this chapter we make the following assumptions.

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

Exercise 6.2. Show how to transform a MCNFP on a digraph with pairs of reverse arcs into a MCNFP on a digraph with no pairs of reverse arcs. The number of arcs and nodes should asymptotically remain the same.

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

$$\begin{aligned} \sum_{e \in \delta^{in}(i)} f(e) - \sum_{e \in \delta^{out}(i)} f(e) = e(i) & \quad \text{for all } i \in V, \\ 0 \leqslant f(e) \leqslant u(e) & \quad \text{for all } e \in A. \end{aligned}$$

- 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 6.3. Prove that $\sum_{i \in V} e(i) = 0$ holds and thus that a feasible flow only exists if the sum of the $b(i)$ is equal to zero.

Let $\mathscr{P}$ be the collection of directed paths of D and let $\mathscr{C}$ be the collection of directed cycles of D. A path-flow is a function $\beta : \mathscr{P} \cup \mathscr{C} \to \mathbb{R}_{\geqslant 0}$ 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 $C \in \mathscr{C}$ let $\delta_{(i,j)}(C)$ be 1 if $(i, j) \in C$ and 0 otherwise.

A path-flow β determines a unique arc-flow

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

Theorem 6.10. *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 arcs with strictly positive flow.

In the first case, let $P = i_0,\dots,i_k$ 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-arcs left.

Each time a path or a cycle is identified, the excess/deficit of some node is set to zero or some arc is set to zero. This implies that we decompose into at most $n+m$ paths and cycles. Since cycle detection sets an arc 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 6.6. *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.

Lemma 6.4. *Suppose that f and f° are feasible flows, then $f - f^\circ$ is a circulation in $D(f^\circ)$. Here $f - f^\circ$ is the flow*

$$(f - f^\circ)(e) = \begin{cases} \max\{0, f(e) - f^\circ(e)\}, & \textit{if } e \in A(D) \\ \max\{0, f^\circ(e) - f(e)\}, & \textit{if } e^{-1} \in A(D) \\ 0, & \textit{otherwise.} \end{cases}$$

Proof. It is very easy to see that the flow $f - f^\circ$ satisfies the capacity constraints. One also has for each $v \in V$

$$\sum e \in \delta^{out}(v)(f(e) - f^\circ(e)) - \sum_{e \in \delta^{in}(v)} (f(e) - f^\circ(e)) = 0.$$

If a term $(f(e) - f^\circ(e))$ is negative, it is replaced by its absolute value and charged as flow on the arc e^{-1} in $D(f^\circ)$ which leaves its contribution to the sum above invariant. □

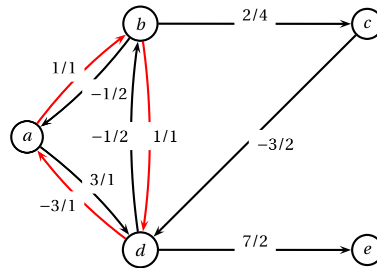


Fig. 6.6: The residual network of the flow in Figure 6.5 and a negative cycle marked by the red edges.

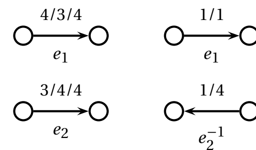


Fig. 6.7: Two arcs $e_1, e_2 \in A$ labeled with $f(e)/f^{\circ}(e)/u(e)$ and the corresponding flow on these arcs (or their reverse) in $D(f^{\circ})$. Arcs in $D(f^{\circ})$ are labeled with flow and capacity values respectively.

Theorem 6.11 (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 6.12 (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. "⇒" 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 in $D(f^\circ)$ with non-positive cost $c^T(f^\circ - f^*) \leqslant 0$. It follows from Theorem 6.11 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. □

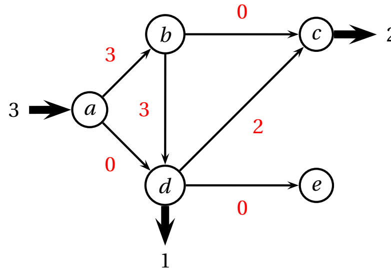


Fig. 6.8: The result of augmenting a flow of one along the negative cycle in Figure 6.6. This flow has cost 12 but is not optimal, since the residual network still contains a negative cycle.

Algorithm 6.1 (Cycle Canceling Algorithm).

1. establish a feasible flow f in the network
2. `WHILE` $D(f)$ contains a negative cycle
   a. detect a negative cycle C in $D(f)$
   b. $\delta = \min\{r(i,j) \mid (i,j) \in C\}$
   c. augment δ units of flow along the cycle C
   d. update $D(f)$
3. `RETURN` f

Theorem 6.13. *The cycle canceling algorithm terminates after a finite number of steps if the MCNFP has an optimal solution.*

Proof. The cycle canceling algorithm reduces the cost in each iteration. We have assumed that the input data is integral. Thus the cost decreases by at least one unit each iteration. Therefore the number of iterations is finite. □

Corollary 6.7. *If the capacities are integral and if the MCNFP has a optimal flow, then it has an optimal flow with integer values only.*

Let $\pi : V \to \mathbb{R}$ be a function (*node potential*). The *reduced cost* of an arc (i,j) w.r.t. π is $c_\pi((i,j)) = c((i,j)) + \pi(i) - \pi(j)$. The potential π is called *feasible* if $c_\pi((i,j)) \geqslant 0$ for all arcs $(i,j) \in A$.

Lemma 6.5. *Let $D = (V,A)$ be a digraph with arc weights $c : A \to \mathbb{R}$. Then D does not have a negative cycle if and only if there exists a feasible node potential π of D with arc weights c.*

Proof. Consider a directed path $P = i_0, i_1, \dots, i_k$. The cost of this path is

$$c(P) = \sum_{j=1}^{k} c((i_{j-1}, i_j)).$$

The reduced cost of this path is equal to

$$c_\pi(P) = \sum_{j=1}^{k} c((i_{j-1}, i_j)) + \pi(i_0) - \pi(i_k).$$

If P is a cycle, then i_0 and i_k are equal, which means that its cost and reduced cost coincide. Thus, if there exists a feasible node potential, then there does not exist a negative cycle.

On the other hand, suppose that D, c does not contain a negative cycle. Add a vertex s to D and the arcs (s,i) for all $i \in V$. The weights (costs) of all these new arcs is 0. Notice that in this way, no new cycles are created, thus still there does not exist a negative cycle. This means we can compute the shortest paths from s to all other nodes $i \in V$. Let π be the function which assigns these shortest paths lengths. Clearly $c_\pi((i,j)) = \pi(i) - \pi(j) + c((i,j)) \geqslant 0$, since the shortest-path length to j is at most the shortest-path length to $i + c((i,j))$. $\square$

This means that we have again a nice way to prove that a flow is optimal. Simply equip the residual network with a feasible node potential.

Corollary 6.8 (Reduced Cost Optimality Condition). *A feasible flow f^* is optimal if and only if there exists a node potential π such that the reduced costs $c_\pi(i,j)$ of each arch (i,j) of $D(f)$ are nonnegative.*

The cycle canceling algorithm is only pseudopolynomial. If we could always chose a minimum cycle (cycle with best improvement) as an augmenting cycle, we would have a polynomial number of iterations. Finding minimum cycles is NP-hard. Instead we augment along *minimum mean cycles.* One can find minimum mean cycles in polynomial time.

The *mean cost* of a cycle $C \in \mathscr{C}$ is the cost of C divided by the number of arcs in C:

$$(\sum_{(i,j)\in C} c(i,j))/|C|.$$

Algorithm 6.2 (Minimum Mean Cycle Canceling, MMCC).

1. establish a feasible flow f in the network
2. `WHILE` $D(f)$ contains a negative cycle
   a. detect a minimum mean cycle C in $D(f)$
   b. $\delta = \min\{r(i,j) \mid (i,j) \in C\}$
   c. augment δ units of flow along the cycle C
   d. update $D(f)$
3. `RETURN` f

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

Lemma 6.6 (See Korte & Vygen [8]). *Let $f_1, f_2, \dots$ 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 arcs.*

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 arcs. 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 arcs 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 arcs for each i, $k < i < l$.

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

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

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

Corollary 6.9. *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 arc of the cycle, which is reverse to one of the succeeding $m-1$ cycles, because every iteration, one arc 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 6.10. *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 6.14 (See Korte & Vygen [8]). *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 arc is *fixed*, which means that the flow through this arc 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 6.9 that

$$\mu(f_1) \leqslant 2\,n\,\mu(f_2) \tag{6.9}$$

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 6.5

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

$$c_\pi(e) \geqslant \mu(f_2), \text{ for all } e \in A(D(f_2)). \tag{6.10}$$

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

$$c_\pi(C) = c(C) = \mu(f_1)\,|C| \leqslant 2\,n\,\mu(f_2)|C|. \tag{6.11}$$

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

$$c_\pi(e_0) \leqslant 2\,n\,\mu(f_2) \tag{6.12}$$

holds. The inequalities (6.10) 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 6.6 that e_0 cannot be anymore in the residual network of a flow after f_2. Thus the flow along the arc 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. (6.12)). Using (6.10) one obtains

$$\begin{aligned} c(C) &= \sum_{e \in C} c_\pi(e) && (6.13)\\ &\geqslant -2\,n\,\mu(f_2) + (n-1)\mu(f_2) && (6.14)\\ &= -(n+1)\,\mu(f_2) && (6.15)\\ &> -n\,\mu(f_2). && (6.16) \end{aligned}$$

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 $\mu(f') \leqslant \mu(f_2)$. □

6.7 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 $C \in \mathscr{C}$ with minimum ratio

$$\frac{c(C)}{p(C)}. \tag{6.17}$$

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

$$\beta \leqslant \frac{c(C)}{p(C)}, \text{ for all } C \in \mathscr{C}. \tag{6.18}$$

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

$$c(C) - \beta\, p(C) \geqslant 0 \text{ for all } C \in \mathscr{C}. \tag{6.19}$$

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 arc 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

$$|c(C_1)/p(C_1) - c(C_2)/p(C_2)| = \left|\frac{c(C_1)\,p(C_2) - c(C_2)\,p(C_1)}{(p(C_1)\,p(C_2))}\right| \tag{6.20}$$

$$\geqslant 1/(n^2 p_{max}^2). \tag{6.21}$$

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 $c \in \mathscr{C}$ with $c(C)/p(C) \in [L, U]$.

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

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

and

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

Let C be a minimum weight cycle w.r.t. the arc 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 $2c$, 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

$$(1/2)^i c \leqslant 1/(n^2 p_{max}^2). \tag{6.22}$$

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 6.15 (Lawler [9]). *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"!

6.7.1 Parametric search

Let us first roughly describe the idea on how to obtain a strongly polynomial algorithm, see [13]. 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

$$f_{i+1}(v)[\beta] = \min\{f_i(v)[\beta], \min_{(u,v)\in A}\{f_i(u)[\beta] + c(u,v) - \beta \cdot p(u,v)\}\}. \tag{6.23}$$

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 arc 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 $\beta_1, \dots, \beta_k$ 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 $\beta_1, \dots, \beta_{\lfloor k/2 \rfloor - 1}$. Otherwise, we replace U by β_{trial} and delete $\beta_{\lfloor k/2 \rfloor + 1}, \dots, b_k$. 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 arc 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

$$O\big(n \cdot (m \cdot n \log n)\big).$$

The total running time is thus

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

Theorem 6.16. *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 $O(n^3 \cdot m \cdot \log n)$.*

6.7.1.1 Exercises

1) Show that there are no two different paths from r to another node in a directed tree $T = (V, A)$.
2) Prove Lemma 6.3.
3) Why can we assume without loss of generality that a minimum cost network has a path from i to j for all $i \neq j \in V$ which is incapacitated?
4) Provide an example of a MCNFP for which the simple cycle-canceling algorithm from above can require an exponential number of cancels, if the cycles are chosen in a disadvantageous way.
5) Provide a proof of Theorem 6.7.
6) Let $Q =< u_1, \ldots, u_k >$ be the queue before an iteration of the **while** loop of the breadth-first-search algorithm. Show that $D[u_i]$ is monotonously increasing and that $D[u_1] + 1 \geqslant D[u_k]$. Conclude that the sequence of assigned labels (over time) is a monotonously increasing sequence.

References

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. F. Eisenbrand, A. Karrenbauer, and C. Xu. Algorithms for longer oled lifetime. In C. Demetrescu, editor, *6th International Workshop on Experimental Algorithms, WEA07*, volume 4525 of *Lecture Notes in Computer Science*, pages 338–351. Springer, 2007.
4. M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer, 1988.
5. L. Khachiyan. A polynomial algorithm in linear programming. *Doklady Akademii Nauk SSSR*, 244:1093–1097, 1979.
6. V. Klee and G. J. Minty. How good is the simplex algorithm? In *Inequalities, III (Proc. Third Sympos., Univ. California, Los Angeles, Calif., 1969; dedicated to the memory of Theodore S. Motzkin)*, pages 159–175. Academic Press, New York, 1972.
7. T. Koch. *Rapid Mathematical Programming*. PhD thesis, Technische Universität Berlin, 2004. ZIB-Report 04-58.
8. B. Korte and J. Vygen. *Combinatorial optimization*, volume 21 of *Algorithms and Combinatorics*. Springer-Verlag, Berlin, second edition, 2002. Theory and algorithms.
9. E. L. Lawler. *Combinatorial optimization: networks and matroids*. Holt, Rinehart and Winston, New York, 1976.
10. L. Lovász. Graph theory and integer programming. *Annals of Discrete Mathematics*, 4:141–158, 1979.
11. J. E. Marsden and M. J. Hoffman. *Elementary Classical Analysis*. Freeman, 2 edition, 1993.
12. J. Matouek and B. Gärtner. *Understanding and Using Linear Programming (Universitext)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.
13. N. Megiddo. Combinatorial optimization with rational objective functions. *Math. Oper. Res.*, 4(4):414–424, 1979.
14. A. S. Nemirovskiy and D. B. Yudin. Informational complexity of mathematical programming. *Izvestiya Akademii Nauk SSSR. Tekhnicheskaya Kibernetika*, (1):88–117, 1983.
15. A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley, 1986.
16. N. Z. Shor. Cut-off method with space extension in convex programming problems. *Cybernetics and systems analysis*, 13(1):94–96, 1977.