Chapter 7

Flow Problems

7.1 Introduction and definitions

Problems related to transport have been investigated since the early fifties. The problem is to route some goods, called *commodities*, from production sites to consumption sites, through a network consisting of communication links inter-connecting the sites (pipe-lines, routes, telecommunication networks). Moreover, each link has a maximum admissible throughput, called the *capacity* of the link.

In general, most of the sites of the network do not produce nor consume anything, they are only used to interconnect links. To each production or consumption site is associated some real number that corresponds to the maximum production or consumption.

The main objective is to maximize the throughput of the traffic.

We first consider networks with directed links, i.e., that can be used in one direction. Formally, a *flow network* is defined as follows.

Definition 7.1 (Flow network)**.** A *flow network* is a four-tuple $(G, pr_{max}, co_{max}, c)$ such that:

- G is a digraph (or a mulitdigraph), the vertices of which represent the sites and the arcs represent the links;
- pr_{max} is a function from $V(G)$ to $\mathbb{R}^+ \cup +\infty$; $pr_{max}(v)$ corresponds to the maximum production possible in v. If v is not a production site, then $pr_{max}(v) = 0$.
- co_{max} is a function from $V(G)$ to $\mathbb{R}^+ \cup +\infty$; $co_{max}(v)$ corresponds to the maximum consumption possible in v. If v is not a consumption site, then $co_{max}(v) = 0$.
- c is a function from $E(G)$ to $\mathbb{R}^+ \cup +\infty$; $c(e)$ corresponds to the capacity of the link e.

Definition 7.2 (Flow)**.** The *flow* is a triple $F = (pr, co, f)$ where

- pr is a function of production such that, for any vertex v, $0 \leq pr(v) \leq pr_{max}(v)$;

- co is a function of consumption such that, for any vertex v, $0 \leq co(v) \leq co_{max}(v)$;

- f is a function over E, called *flow function* that satisfies the following constraints:

Positivity:	$\forall e \in E(G)$,	$f(e) \geq 0$
Capacity constraint:	$\forall e \in E(G)$,	$f(e) \leq c(e)$
Flow conservation:	$\forall v \in V(G)$,	$\sum_{(u,v)\in E(G)} f((u,v)) + pr(v) = \sum_{(v,u)\in E(G)} f((v,u)) + co(v)$

By summing, the $|V(G)|$ equations of flow conservation, we get:

$$\sum_{v\in V(G)} pr(v) = \sum_{v\in V(G)} co(v)$$

In other words, the sum of all produced commodities equals the sum of all consumed commodities. This value corresponds to the amount of routed traffic, it is the *flow value*, denoted by $v(F)$.

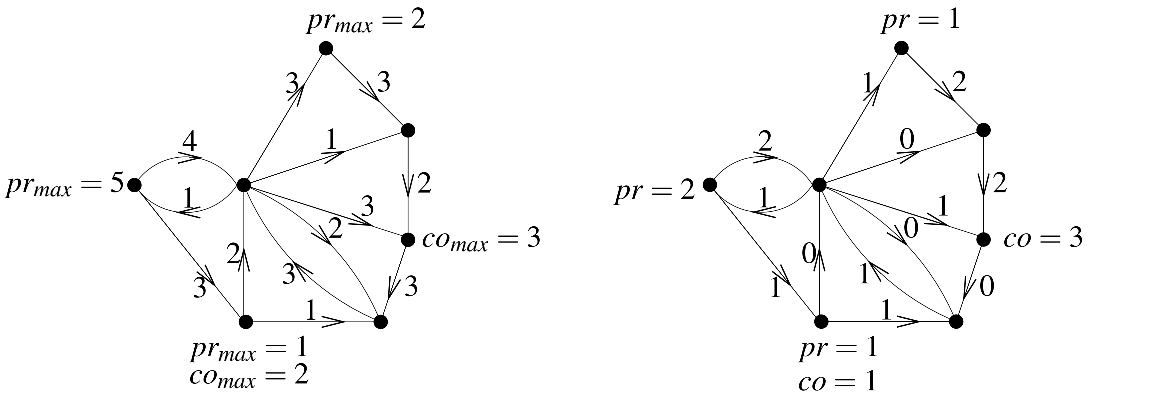


Figure 7.1: Example of flow network (left) and a flow of value 4 in it (right)

Hence, the problem is the following:

Problem 7.3 (Maximum Flow)**.**
Instance: a flow network N.
Find: a flow with maximum value.

A flow with maximum value is said to be a *maximum flow*.

7.2 Reducing to an elementary network

Before studying this problem, we show that it is equivalent to consider an elementary problem where there is a unique production site with infinite maximum production and a unique consumption site with infinite maximum consumption.

Definition 7.4 (Elementary network). A flow network $(G, \infty_s, \infty_p, c)$ is *elementary*:

- s and p are two distinct vertices where s is a *source* (i. e. $d^-(s) = 0$) and p is a *sink* (i.e. $d^+(p) = 0$);
- $\infty_s(s) = +\infty$ and $\infty_s(v) = 0$ for all $v \neq s$;
- $\infty_p(p) = +\infty$ and $\infty_p(v) = 0$ for all $v \neq p$.

We denote $(G, \infty_s, \infty_p, c)$ by (G, s, p, c).

Definition 7.5 (Associated elementary network). If $N = (G, pr_{max}, co_{max}, c)$ is a flow network, its *associated* elementary network is the following elementary network $N = (\bar{G}, s, p, \bar{c})$ obtained from N in the following way:

- add a source s and a sink p;
- link s to all vertices v with non-null production with an arc (s, v) of capacity $pr_{max}(v)$;
- link all vertices v with non-null consumption to p with a link (v, p) of capacity $co_{max}(v)$;

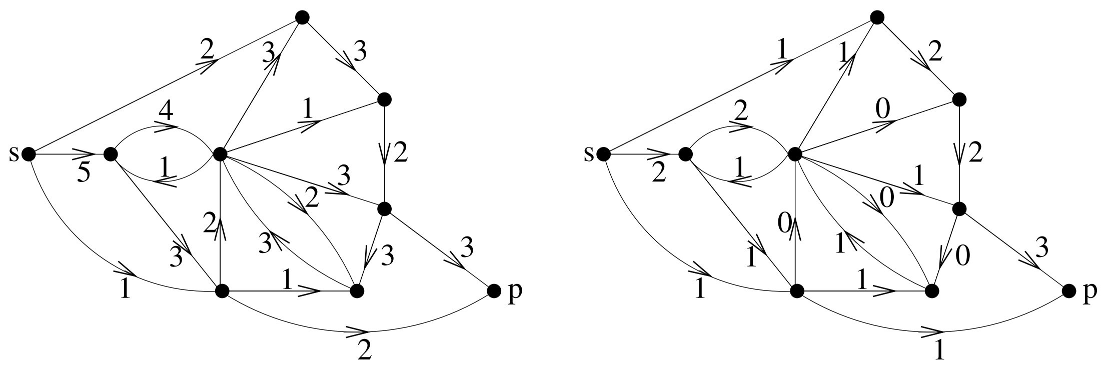


Figure 7.2: Elementary network associated to the flow network depicted in Figure 7.1 (right) and the flow in this network corresponding to the one of Figure 7.1 (left)

It is easy to see that there is a one-to-one correspondence between any flow $F = (pr, co, f)$ of the elementary network and a flow $F' = (pr', co', f')$ of the initial problem defined as follows: for any production site u, $pr'(u) = f((s, u))$; for any consumption site v, $co'(v) = f((v, p))$ and for any arc (x, y), $f'((x, y)) = f((x, y))$.

Clearly, both flows have same value $v(F) = v(F') = pr(s) = co(p)$. Hence:
Finding a maximum flow in a flow network is equivalent to finding a maximum flow solve in its associated elementary network.

In the following, we only consider elementary flow networks.

Note that, in an (elementary) flow network $N = (G, s, p, c)$, the flow conservation constraint can be written:

$$\forall v \in V(G) \setminus \{s,p\}, \sum_{(u,v) \in E(G)} f((u,v)) = \sum_{(v,u) \in E(G)} f((v,u))$$

Moreover, a flow $F = (pr, co, f)$ of an (elementary) flow network is well defined by the flow function f since, by the flow conservation constraint in s and p, we have

$$\begin{array}{rcl} v(F) = pr(s) & = & \sum_{u \in V(G), (s,u) \in E(G)} f((s,u)) \\ = co(p) & = & \sum_{u \in V(G), (u,p) \in E(G)} f((u,p)) \end{array}$$

Therefore, for ease of presentation, we often identify a flow F with its function f, and we note the flow value by $v(f)$.

To simplify the notations in the sequel, for a flow f or a capacity c, and an arc (u,v), we write $f(u,v)$ instead of $f((u,v))$, and $c(u,v)$ instead of $c((u,v))$.

7.3 Cut and upper bound on the maximum flow value

We will show that the value of a maximum flow in a network is limited by the existence of some bottlenecks through which the traffic must go. Roughly, to go from the source to the sink, the flow must cross the border of a set of vertices, its size (the sum of the capacities of the corresponding edges) will limit the value of the flow. Such a border is called a *cut*.

Definition 7.6 (Cut)**.** In a flow network $N = (G, s, p, c)$, an (s, p)*-cut*, or simply a *cut*, is a bipartition $C = (V_s, V_p)$ of the vertices of G such that $s \in V_s$ and $p \in V_p$. The arcs form V_s to V_p (i.e. with tail in V_s and head in V_p) are the *arcs of* C. Their set is denoted by $E(C)$. The *capacity* of the cut C, denoted by $\delta(C)$, is the sum of the capacities of its arcs: $\sum_{e \in E(C)} c(e)$.

Let f be a flow and $C = (V_s, V_p)$ be a cut, $out(f, C)$ denotes the flow on arcs leaving V_s and $in(f, C)$ the flow entering V_s :

$$\begin{array}{rcl} out(f,C) & = & \sum_{(u,v) \in E(C),\ u \in V_s, v \in V_p} f(u,v) \\ in(f,C) & = & \sum_{(u,v) \in E(G),\ u \in V_p, v \in V_s} f(u,v) \end{array}$$

Note that the flow conservation implies that

$$\text{for all cut } C, \quad v(f) = out(f,C) - in(f,C)$$

In particular, the value of the flow is always at most $out(f, C)$. But clearly $out(f, C) \leq \delta(C)$,so

$$v(f) \leq \delta(C). \tag{7.1}$$

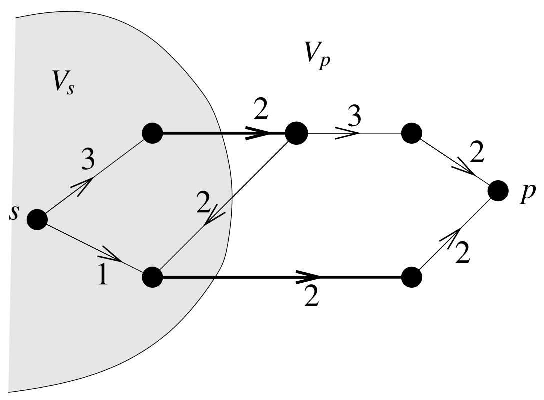


Figure 7.3: A flow network and a cut with capacity 4. Bold arcs are those of the cut

A cut can be viewed as a set of arcs that the flow must cross and whose capacity limits the value of the flow.

Let $v_{max} = max\{v(f), f$ a flow$\}$ and $\delta_{min} = min\{\delta(C),\ C$ a cut$\}$. Equation 7.1 applied to a maximum flow and a minimum-capacity cut yields

$$v_{max} \leq \delta_{min}. \tag{7.2}$$

In fact, the next theorem,due to Ford and Fulkerson, states that there is equality.

Theorem 7.7 (Ford & Fulkerson)**.** *The maximum value of a flow equals the minimum capacity of a cut, i.e.,*

$$v_{max} = \delta_{min}.$$

We often say that *max flow equals min cut*. It is an example of“min-max theorem”

There are many proofs of this theorem, some being non-constructive proof of this theorem. In the next section, we present the original proof which is based on an algorithm computing a maximum flow and a cut with minimum capacity.

7.4 Auxiliary Network and “push” Algorithm

Most of the algorithms for computing maximum flows are based on the following idea: Starting from an existing flow (initially, it may be null), the flow is increased by going from the source to the sink by “pushing” the commodity where it is possible.

The difference between the algorithms mainly consists of the way used to decide where and how to push some flow.

For this purpose, we define an *auxiliary network*. This graph, denoted by $N(f)$, depends on the existing flow f.

Definition 7.8 (Auxiliary network)**.** Given a flow network $N = (G, s, p, c)$ and a flow f, we build the auxiliary network $N(f) = (G(f), s, p, c_f)$ as follows.
For any pair of vertices (u, v), let

$$c_f(u,v) = c(u,v) - f(u,v) + f(v,u)$$

with $c(u,v)$, $f(u,v)$ and $f(v,u)$ equal to 0 when they are not defined (if (u,v) is not an arc of G). Note that $c_f(u,v) \geq 0$ since $c(u,v) - f(u,v) \geq 0$. Then, $G(f)$ is defined as follows

$$\begin{array}{rcl} V(G(f)) & = & V(G) \\ E(G(f)) & = & \{(u,v) \mid c_f(u,v) > 0\} \end{array}$$

Note that $c_f(u,v) + c_f(v,u) = c(u,v) + c(v,u)$. Intuitively, $c_f(u,v)$ is the sum of the remaining capacity on the arc (u,v), i.e., $c(u,v) - f(u,v)$, plus a virtual capacity $f(v,u)$, that allows to "remove" some flow on the arc (v,u), which corresponds to virtually push some flow along (u,v). See examples in Figures 7.4 and 7.5.

Note that the auxiliary network has no arc with capacity 0. This is important because it ensures that, for any directed path P in $G(f)$, the minimum capacity of the arcs of P is positive.

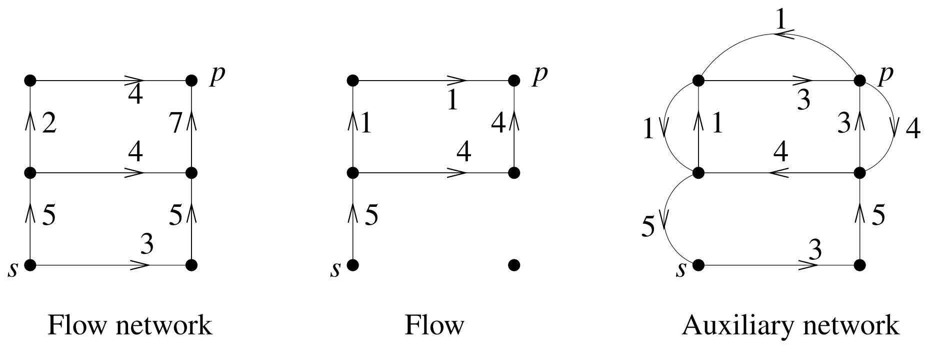


Figure 7.4: A flow network, a flow and the corresponding auxiliary network

Let P be a directed (s, p)-path in $N(f)$ where the minimum capacity of an arc in P is $\varepsilon > 0$. The flow f' obtained by *pushing* ε *units of flow* along P is defined by:
For any arc $(u,v) \in E(P)$, we push the flow along (u,v), that is, we increase the flow with ε units on (u,v). Two cases may happen:

- If the remaining capacity of (u,v) is sufficient for the ε units of flow, i.e., $f(u,v) + \varepsilon \leq c(u,v)$, then $f'(u,v) = f(u,v) + \varepsilon$ and $f'(v,u) = f(v,u)$.

- If the remaining capacity of (u,v) is not sufficient for the ε units of flow, i.e., $f(u,v) + \varepsilon > c(u,v)$, then, by definition of the auxiliary network, we have $f(v,u) \geq f(u,v) + \varepsilon - c(u,v) > 0$. Hence, (v,u) has a flow excess of $f(u,v) + \varepsilon - c(u,v)$ units that we must remove. We set $f'(u,v) = c(u,v)$ and $f'(v,u) = f(v,u) - (f(u,v) + \varepsilon) + c(u,v)$.

If (u,v) and (v,u) do not belong to P, the flow remains unchanged on these arcs, $f'(u,v) = f(u,v)$ and $f'(v,u) = f(v,u)$.

Lemma 7.9. *If f is a flow of value $v(f)$, then f' is a flow of value $v(f)+\varepsilon$.*

Proof. See Exercice 7.5. □

For instance, from the flow depicted in Figure 7.4, by taking the directed (s,p)-path depicted in Figure 7.5 with minimum capacity 1 and pushing the flow, we obtain the new flow and the new auxiliary network depicted in Figure 7.5.

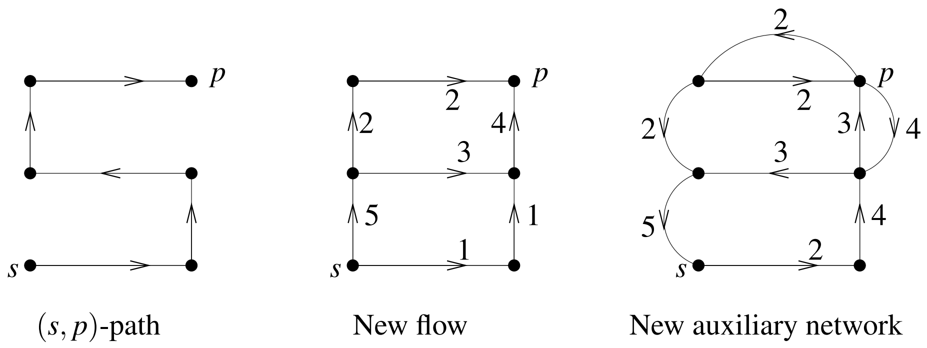


Figure 7.5: Push along a directed (s,p)-path of the auxiliary network in Figure 7.4.

Now, we have some elements of an algorithm:

1. Start with any flow f;
2. Compute the auxiliary network $N(f)$;
3. Find a directed path from s to p in $G(f)$;
4. If such a directed path P exists, then push some flow along P and update f.

Note that, if there is no directed path from the source to the sink in the auxiliary graph, the previous algorithm does nothing. We will see that, in this case, the existing flow is maximum.

Proposition 7.10. *If $G(f)$ does not contain a path from the source to the sink, then the flow f is maximum.*

Proof. Let V_s be the set of all vertices that can be reached from s in $G(f)$. V_s does not contain the sink since there are no (s,p)-paths. Hence, $p \in V_p = V \setminus V_s$. Let C be the cut (V_s,V_p).

Let (u,v) be an arc of C. By definition of V_s, the arc (u,v) is not in $G(f)$. So, by definition of $G(f)$, this means that $c_f(u,v) = 0$. Hence, f is such that $f(u,v) = c(u,v)$ and $f(v,u) = 0$.

We get that $out(f,C) = \delta(C)$ and $in(f,C) = 0$. Informally, there is no flow entering V_s and all arcs of C (that is leaving V_s) are saturated.

But $v(f) = out(f,C) - in(f,C)$, therefore

$$v(f) = \delta(C)$$

The current flow has the same value as the capacity of the cut C. Hence, f is maximum by Equation (7.2). □

Note that this proof also exhibit a cut (V_s,V_p) with same capacity as the maximum value of a flow. Hence it is a minimum-capacity cut. It shows that it is easy to find a minimum-capacity cut (V_s,V_p) from a maximum flow: V_s is the set of the vertices reachable from the source in the auxiliary network and V_p its complement.

Now, we can prove Theorem 7.7.

Proof of Theorem 7.7: Let f be a flow with maximum value in a flow network (G,s,p,c), and let $N(f)$ be the auxiliary network. In $G(f)$, there are no directed (s,p)-paths, otherwise we obtain a flow with greater value by pushing some flow along this path (Lemma 7.9). Hence, by the proof of Proposition 7.10, the cut $C = (V_s,V_p)$, where V_s is the set of vertices v such that there is a directed (s,v)-path in $G(f)$, has capacity $v(f) = v_{max}$. Therefore, $\delta_{min} \leq \delta(C) \leq v_{max}$.
□

7.5 Ford-Fulkerson algorithm

We now have the following algorithm:

Algorithm 7.1 (Ford and Fulkerson (1956))**.**

1. Start with null flow $f = 0$;
2. Compute the auxiliary network $N(f)$;
3. Look for a directed path from s to p in $G(f)$;
4. If such a directed path P exists, then push some flow along P, update f and go to 2;
5. Else terminate and return f.

This algorithm is correct in the sense that If the algorithms terminates, then it returns a maximum solution. But

1) does it always terminate?

2) If it terminates, what is its complexity?

The answer to question 1 is somehow yes and no: we will see that the algorithm always terminates if the capacities are rational, but it can take infinite time if capacities are real. Besides, even when the capacities are integers, the running time depends linearly on the value of the maximum flow that may be huge.

Analysis of Ford-Fulkerson Algorithm

Proposition 7.11. *(i) If all capacities are integers, then Algorithm 7.1 terminates after at most v_{max} searches of a directed path.*

(ii) If all the capacities are rational, then Algorithm 7.1 terminates after at most $\mu \cdot v_{max}$ searches of a directed path, with μ the least common multiple of the denominators of the capacities.

Proof. (i) If capacities are integers, at each iteration, the algorithm pushes at least one unit of flow (since, in this case, ε is integral). Each iteration requires the search of a path from s to p in $G(f)$. This can be done in time $O(|E(G)|)$ by any search algorithm (See Chapter 2). Hence, its total running time is at most

$$O(\ v_{max} \cdot |E(G)|).$$

(ii) If capacities are rational, the algorithm terminates since the flow increases of at least $\frac{1}{\mu}$ at each iteration. μ can be very large, but it is fixed. Actually, we can solve the problem by multiplying all capacities by μ and by solving the integral problem obtained: it is proportional the initial one.

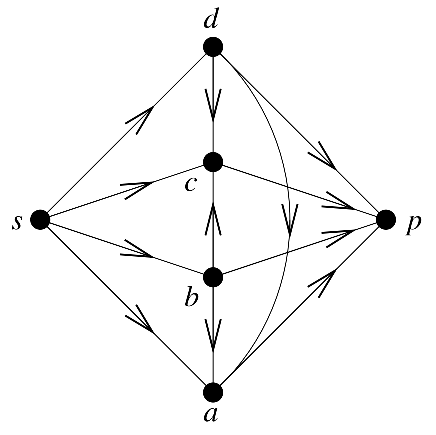
□

Remark 7.12. If some capacities are integers, then the Ford-Fulkerson algorithm returns a maximum flow with integral value.

Proposition 7.13. *If the capacities are real, the algorithm may not terminate. Moreover, the increasing sequence of the flow value may converge to a value a lot smaller than the optimal value.*

Proof. Consider the flow network shown in Figure 7.6 for which the capacities of all edges are infinite (or if the reader prefers, a huge integer M). Let α denote the positive root of $x^3 + x - 1 = 0$. Clearly $1/2 < \alpha < 1$. Let the initial flow f_0 be as shown in Figure 7.6.

We shall prove that by employing a well (or very badly) chosen sequence of four augmenting paths over and over, Algorithm 7.1 will produce an infinite sequence of flows, the values of which are monotone increasing and which converge to a limit not exceeding 16. For any $m \geq 0$, start from the flow f_{4m} and push along the directed path (s,c,d,a,b,p). The pushed amount is α^{4m+1} because of arc (a,b) in the auxiliary network. The resulting flow is f_{4m+1}. Push along the directed path (s,c,b,a,d,p) an amount of α^{4m+2} (because of (c,b)) to produce f_{4m+2}, push along the directed path (s,a,b,c,d,p) an amount of α^{4m+3} (because of (c,d)) to produce



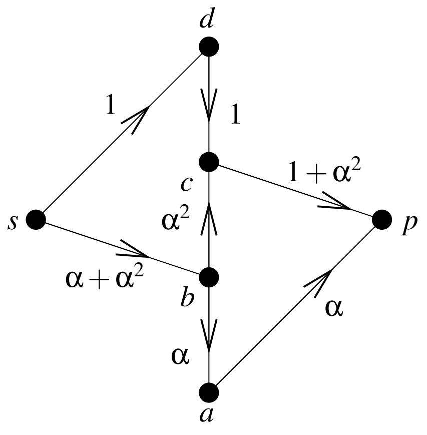


Figure 7.6: A flow network and the initial flow f_0

f_{4m+3} and push along the directed path (s,a,d,c,b,p) an amount of α^{4m+4} (because of (a,d)) to produce f_{4m+4}. See Figure 7.7.

For $k \geq 1$, the augmentation of the value from f_{k-1} to f_k is α^k and hence

$$\begin{aligned}
v(f_k) &= v(f_0)+\alpha+\alpha^2+\cdots+\alpha^r \\
&= (1+\alpha+\alpha^2)+\alpha+\alpha^2+\cdots+\alpha^r \\
&= \frac{1}{\alpha}(\alpha+\alpha^2+\alpha^3+\alpha^2+\alpha^3+\cdots+\alpha^{r+1}) \\
&= \frac{1}{\alpha}(\alpha+\alpha^2+(1-\alpha)+\alpha^2+\alpha^3+\cdots+\alpha^{r+1}) \\
&= \frac{1}{\alpha}(1+2\alpha^2+\alpha^3+\cdots+\alpha^{r+1}) \\
&< \frac{1}{\alpha}(1+\alpha+\alpha^2+\alpha^3+\cdots+\alpha^{r+1}) \\
&< \frac{1}{\alpha-\alpha^2} = \frac{1}{\alpha^4} < 16.
\end{aligned}$$

From this construction, it is easy to construct an example of a network where some capacities are irrational such that even starting with a null flow, the sequence of flows obtained by pushing along some directed paths converges to a value less than 16, while the value of a maximum flow is infinite (or arbitrarily large). See Exercise 7.9.

□

Remark 7.14. While the termination is not ensured in case of irrational capacities, Theorem 7.7 ($v_{max} = \delta_{min}$) remains valid. We prove the real case by taking the limit of the rational case. In practice, the rational case is the only important problem, since computers works with a finite precision.

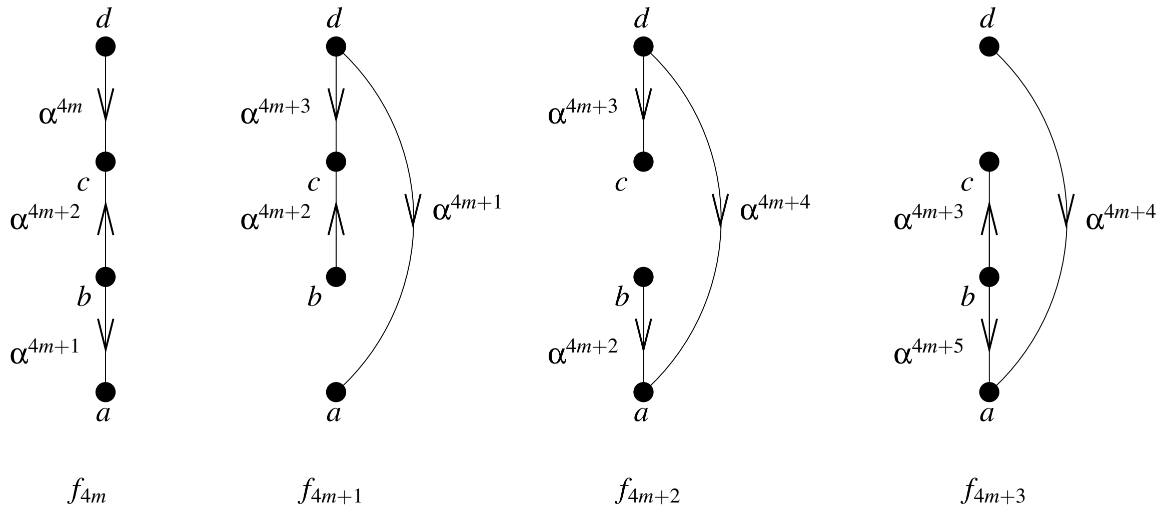


Figure 7.7: The sequence of flows. The arcs leaving s and the arcs entering p are not drawn

The bound of Proposition 7.11 on the number of pushes is not good because it depends on the value of the maximum flow which may be huge. Figure 7.8 shows an example where v_{max} pushes, if they are badly chosen, are necessary to reach a flow with maximum value. Indeed, if

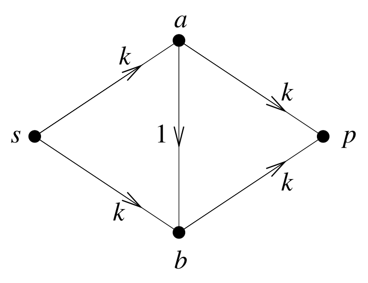


Figure 7.8: Example where $2k = v_{max}$ pushes may be performed.

the pushes are alternatively performed along $P_1 = (s,a,b,p)$ and $P_2 = (s,b,a,p)$, then one unit of flow is pushed at each iteration because the capacity of (a,b) or (b,a) equals 1. Therefore, $2k = v_{max}$ pushes are necessary (k along P_1 and k along P_2).

To improve the previous algorithm and to ensure its quick termination in all cases, the idea is to consider the pushes on some specific directed paths.

7.6 Pushing along shortest paths

Instead of pushing the flow along an arbitrary directed path (s,p)-path in the auxiliary network, a better algorithm consists in pushing along a shortest directed (s,p)-path. Here, by shortest path, we mean a shortest path in terms of number of arcs, not of total capacity.

Algorithm 7.2 (Edmonds-Karp, 1970)**.**

1. Start with null flow $f = 0$;
2. Compute the auxiliary network $N(f)$;
3. Look for a shortest directed path from s to p in $G(f)$;
4. If such a path P exists, push some flow along P, update f and go to 2;
5. Else terminate and return f.

We will show that such an algorithm performs at most $\frac{|E(G)||V(G)|}{2}$ iterations. That is, its time complexity is at most $\frac{|E(G)||V(G)|}{2}$ times the complexity of finding a shortest path. Note that, contrary to the Ford-Fulkerson Algorithm (7.1), this bound is independent from the capacities.

The proof is based on two simple properties.

1) During the iterations, the distance between s and p in the auxiliary graph cannot decrease.

2) During the iterations, the distance between s and p remains unchanged at most $|E(G)|$ consecutive iterations.

Let f_0 be a flow and let $G_0 = G(f_0)$; We set:

- E' the set of arcs of G_0 that belong to a shortest path from s to p in G_0;
- E'^- the arcs obtained by reversing the ones in E' ;
- G_1 the graph obtained from G_0 by adding the arcs in E'^-.

Lemma 7.15. *The graph G_1 has the following properties:*

(i) A directed path from s to p with length $dist_{G_0}(s,p)$ does not use any arc in E'^-.

(ii) The distance from s to p in G_1 is $dist_{G_0}(s,p)$;

Proof. (i) Let P be a directed (s,p)-path in G_1 that contains some arcs in E'^-, and let (v,u) be the last arc of E'^- that belongs to P. Let us show that P is not a shortest directed (s,p)-path. Since (u,v) belongs to a shortest directed (s,p)-path, we have $dist_{G_0}(u,p) = dist_{G_0}(v,p)+1$. Since between u and p, the path P uses only arcs in G_0, its length is $l+1+dist_{G_0}(u,p)$ (where l is the length of the subpath of P between s and v). Hence, P has length $l+2+dist_{G_0}(v,p)$. The path P' obtained by following P until v and then using a shortest directed (v,p)-path has length $l+dist_{G_0}(v,p) = l+dist_{G_0}(v,p)$. Therefore, P is not a shortest directed (s,p)-path.

(ii) Follows from (i). □

Theorem 7.16. *Algorithm 7.2 performs at most $|E(G)||V(G)|$ iterations. Its time-complexity is $|E(G)||V(G)|$ searches of shortest directed paths, so $O(|E(G)|^2|V(G)|)$.*

Proof. During a push, we add to the auxiliary network the arcs in E'^-. By Lemma 7.15, it does not decrease the distance between s and p. The algorithm consists of $|V|$ steps corresponding to the set of all possible distances between s and p. A step consists of a set of iterations when the distance between s and p remains unchanged.

Let us consider successive iterations that let the distance $dist(s,p)$ unchanged. During some iterations, the auxiliary network G_0 changes since some arcs are added and other arcs are removed. However, since $d(s,p)$ remains the same, the paths along which pushes are performed use only arcs of G_0. Since at each iteration, at least one arc of G_0 is removed (we push the maximum possible along the chosen path, so one arc is removed), at most $|E(G)|$ such iterations are performed.

Since $dist(s,p) \leq |V(G)|$, we have at most $|V(G)|$ steps of at most $|E(G)|$ iterations each. At each iteration we mainly have to find a shortest directed (s,p)-path. This can be done in time $O(|E(G)|)$ by any search algorithm (See Chapter 2). Hence the total complexity is $O(|E(G)|^2|V(G)|)$.

□

7.7 Algorithm using a scale factor

If the capacities are integral then Algorithm 7.1 may take time to find the maximum flow becasue of the disparity of the values of the capacities. In the example in Figure 7.8, the pushes may be performed along an arc with capacity 1, while there is a path between the source and the sink with capacity k. This problem may be overcome by using a scale factor. The idea is to try to work with capacities with similar values, to saturate them and then to consider capacities with smaller values. The idea is to start with the greatest capacities, to push some flow in a network that consists only of these links and to try to saturate them as much as possible. Then, the network is replaced by the current auxiliary network to which we add new links with smaller capacities than the one considered yet.

Algorithm 7.3 (Scaling Algorithm).

```
0. Compute the smallest integer $m$ such that $c(e) < 2^m$ for any arc $e$.
   For all $e \in E(G)$, set $c(e) = \sum_{j=0}^{m-1} 2^j c_j(e)$ with $c_j(e) \in \{0,1\}$.
1. $k := m-1$; for all $e \in E(G)$, $c'(e) := 0$ and $f(e) := 0$;
2. If $k < 0$, then terminate, else for all $e \in E(G)$, $c'(e) := c'(e) + 2^k c_k(e)$.
3. Increase as much as possible the flow $f$ in $(G,s,p,c')$; $k := k-1$; go to Step 2.
```

Clearly, this algorithm computes a maximum flow because, at the end, all capacities are taken into account.

Let us consider the complexity of this algorithm. For this purpose, we need the following proposition the proof of which is left in Exercise 7.10.

Proposition 7.17. *Let $N = (G,s,p,c)$ be a flow network and α a positive real. Let e be an arc of G and N' be the flow network (G,s,p,c') where c' is defined by $c'(e) = c(e) + \alpha$ and $c'(f) = c(f)$ for all $f \neq e$. Then $v_{max}(N') \leq v_{max}(N) + \alpha$.*

Theorem 7.18. *Algorithm 7.3 performs less than $(\log_2(v_{max}) + 1) \cdot |E(G)|$ pushes.*

Proof. Let $c'_i = \sum_{j=i}^{m-1} 2^j c_j$ be the capacity at Step 3 when $k = i$, g_i be the flow added during this step, and f_{i+1} be the total flow before this step. Hence $f_i = f_{i+1} + g_i$.

Let us prove by decreasing induction that at Step 3-(i) Algorithm 7.3 performs at most $|E(G)|$ pushes and that $f_i(e)$ is a multiple of 2^i for every arc e.
The results holds trivially for $i = n$. Suppose now that the results holds for $i+1$. Since the capacity c'_i and the flow f^{i+1} are multiple of 2^i, in the auxiliary network, the capacity of the arcs are multiples of 2^i. Hence finding g_i corresponds to finding a maximum flow g'_i in the network with integral capacities $(G,s,p,(c'_i - f_{i+1})/2^i)$ with $v(g'_i) \leq v(g_i)/2^i$. Moreover, at the end of Step 3-($i+1$), the flow could not be increased anymore, therefore, there are no directed paths with minimum capacity 2^{i+1}. Hence, at Step 3.(i), any directed path in the auxiliary network has minimal capacity 2^i. Hence, each iteration pushes exactly 2^i units of flow at Step 3.(i). Thus $v(g_i) \leq 2^i \cdot |E(G)|$ and so $v(g'_i) \leq 2^i|E(G)|$. By Proposition 7.11, finding g'_i and g_i is done in at most $|E(G)|$ pushes. In addition, by Remark 7.12, the flow g'_i has integral values and so the values of g_i are multiple of 2^i.

Let i_0 be the largest integer such that a push has been done at i_0. If $2^{i_0} \geq v_{max}$, then after pushing once at Step 3-(i_0), we obtained a flow of value 2^{i_0}, which must be a maximum flow.

Otherwise, $i_0 < \log_2(v_{max})$ and for each i, $0 \leq i \leq i_0$, Algorithm 7.3 performs at most $|E(G)|$ pushes. Hence in total, the number of pushes it at most $(i_0+1)|E(G)| \leq (\log_2(v_{max}) + 1) \cdot |E(G)|$. □

7.8 Flows in undirected graphs

Until now, we have considered the problem in directed graphs. However, there are some contexts in which the corresponding network is undirected. For instance, when the links are bidirectional. In this case, each link has one maximum capacity but the flow may circulate in both directions if the sum of the two traffics is at most the capacity of the link.

A *undirected flow network* $N = (G, pr_{max}, co_{max}, c)$ is defined similarly to the direct case. The definition of the flow is a bit modified since two values must be associated to each edge uv: $f(u,v)$ corresponds to the traffic from u to v and $f(v,u)$ corresponds to the traffic from v to u.

Definition 7.19 (Undirected Flow)**.** The *flow* is a three-tuple $F = (pr, co, f)$ where

- pr is a function of production such that, for any vertex v, $0 \leq pr(v) \leq pr_{max}(v)$;
- co is a function of consumption such that, for any vertex v, $0 \leq co(v) \leq co_{max}(v)$;
- f is a function over the ordered pair (u,v) (with u and v adjacent), called *flow function* that satisfies the following constraints:

Positivity:	$\forall e \in E$,	$f(e) \geq 0$
Capacity constraint:	$\forall uv \in E$,	$f((u,v)) + f((v,u)) \leq c(uv)$
Flow conservation:	$\forall v \in V$,	$\sum_{(u,v)\in E} f((u,v)) + pr(v) = \sum_{(v,u)\in E} f((v,u)) + co(v)$

As in the directed case, the *value of the flow F* is

$$v(F) = \sum_{v\in V} pr(v) = \sum_{v\in V} co(v)$$

Let $N = (G, pr_{max}, co_{max}, c)$ be an undirected flow network. The (directed) flow network *associated* to N is $\vec{N} = (\vec{G}, pr_{max}, co_{max}, \vec{c})$ obtained by replacing each edge uv by an arc (u,v) and an arc (v,u) each of which has capacity $c(u,v)$. Formally, $(\vec{G} = (V(G), \bigcup_{uv\in E(G)} \{(u,v),(v,u)\})$ and $\vec{c}((u,v)) = \vec{c}((v,u)) = c(uv)$.

Clearly, a flow function f in N and a flow function $\vec{f}$ in $\vec{N}$ satisfy the same constraints but the capacity constraint. The one in N:

$$\forall uv \in E(G), f((u,v)) + f((v,u)) \leq c(uv)$$

is stronger than the one in $\vec{N}$:

$$\forall uv \in E(G), \vec{f}((u,v)) \leq c(uv) \text{ et } \vec{f}((v,u)) \leq c(uv)$$

Hence, any flow of N is a flow in $\vec{N}$. Hence, the maximum value of a flow in N is at most the maximum value of a flow in $\vec{N}$. In other words, $v_{max}(N) \leq v_{max}(\vec{N})$. We show that they are equal.

Definition 7.20 (simple flow)**.** A flow is *simple* if, for any pair of vertices $\{u,v\}$, we have either $f(u,v) = 0$ or $f(v,u) = 0$. To any flow, there is a corresponding simple flow defined as follows. If $f(u,v) \geq f(v,u) > 0$ then $\tilde{f}(u,v) = f(u,v) - f(v,u)$ and $\tilde{f}(v,u) = 0$. It is easy to see that $v(f) = v(\tilde{f})$ since $f(v,s) = 0$ for any vertex v because $d^-(s) = 0$ and so $f(s,v) = \tilde{f}(s,v)$.

Proposition 7.21. *Let N be an undirected flow network and let $\vec{N}$ be the associated flow network:*

$$v_{max}(N) = v_{max}(\vec{N})$$

Proof. Let $\vec{F} = (pr, co, \vec{f})$ be a flow in $\vec{N}$. Let $F = (pr, co, f)$ be the simple flow of $\vec{F}$. Then $v(F) = v(\vec{F})$. Since F is simple, for any edge $uv \in E(G)$, we have $f(u,v) = 0$ or $f(v,u) = 0$. Besides, the capacity constraint in $\vec{N}$ gives $f(u,v) \leq c(uv)$ ou $f(v,u) \leq c(uv)$. Hence, for any $uv \in E(G)$, $f((u,v)) + f((v,u)) \leq c(uv)$. Then, F is a flow for N.

Therefore, any flow of $\vec{N}$ corresponds to a flow of N with same value. □

This proposition allows us to reduce the undirected problem to the directed case. To find a maximum flow in an undirected network, it is sufficient to solve the problem in the associated directed network and to take the corresponding simple flow.

7.9 Applications of flows

7.9.1 Connectivity in graphs

Menger's Theorem (Theorem 5.18 is very closely related to Theorem 7.7. Observe that the arc set of an (s,p)-cut in a flow network corresponds to an (s,p)-edge-separator. Hence one can deduce Menger's Theorem from Theorem 7.7. We now do it for Theorem 5.18-(ii) for digraphs. In fact the proof of this results we gave in Section 5.6 was using the push technique (in disguise).

Proof of Theorem 5.18-(ii) for digraphs. Let G be a digraph. The edge-connectivity $\kappa'(s,t)$ is the value of the capacity of a cut in the flow network N obtained by assigning to each arc a capacity of 1, and choosing s as source and t as sink. Indeed, if $C=(V_s,V_t)$ is a cut, then after the removal of the arcs of C, there remain no (s,t)-paths. Hence $\kappa'(s,t)\leq\delta_{min}(N)$. Reciprocally, let E' be an arc-separator of G and $G'=G\setminus E'$. Let V_s be the set of vertices w of G' such that there exists a directed (s,w)-path in G', and $V_t=V\setminus V_u$. By definition of V_s, any arc (x,y) with $x\in V_s$ and $y\in V_t$ is in E'. Hence, $\delta((V_s,V_t))\leq|E'|$. Therefore, $\delta_{min}(N)\leq\kappa'(s,t)$.

From Theorem 7.7, there is a flow of value $\kappa'(s,t)$ in this network. Moreover, by Remark 7.12, we may assume that this flow is integral. Since the edges have unit capacity, the flow of value $\kappa'(s,t)$ can be decomposed into a set of $\kappa'(s,t)$ pairwise edge-disjoint (s,t)-paths. Hence $\kappa'(s,t)=\Pi'(s,t)$. □

The other cases of Menger's Theorem may also be derived from Theorem 7.7. See Exercise 7.13. Thus Menger's Theorem can be viewed as a particular case of Theorem 7.7. In fact, they are equivalent and it is not too difficult to prove Theorem 7.7 from Menger's Theorem. See Exercise 7.14.

7.9.2 Maximum matching in bipartite graphs

The theorems on matching in bipartite graphs that we proved in Chapter 6 are direct applications of flows. Indeed to every bipartite graph $G=((A,B),E)$, one can associate the flow network $N_G=(H,s,p,c)$ defined as follows: H is the digraph obtained from G by orienting all edges of G from A to B and adding a source s and a sink p, all arcs (s,a) for $a\in A$ and (b,p) for $b\in B$; the capacity equals 1 for all arcs. See Figure 7.9.

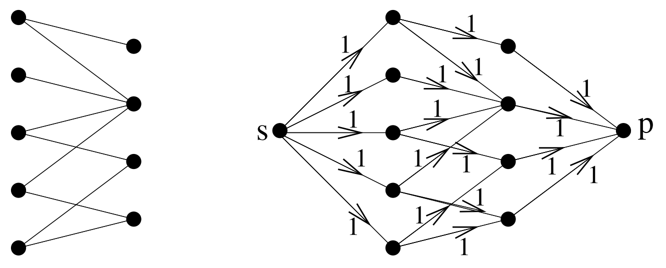


Figure 7.9: A bipartite graph and its associated flow network

There is a one-to-one correspondence between the matchings of G and the integral flows in N_G: to every matching M corresponds the flow f_M with value 1 on the arc set $\bigcup_{(a,b)\in E(G)}\{(s,a),(a,b),(b,p)\}$. Moreover the size of M is equal to the value of the flow f_M. In addition, an M-augmenting path in G corresponds to a directed $(s,p$-path in the auxiliary network $N_G(f_M)$. Hence Algorithm 6.2 is a particular case of Algorithm 7.1.

One can also deduce all the results of Chapter 6 from Theorem 7.7. For example, we now give a proof of Theorem 6.3 (*"Let $G=(A,B)$ bipartite, there is a matching of size k if and only if $\forall S\subset A, |A|-|S|+|N(S)|\geq k$"*) using flows.

Proof of Theorem 6.3. Let us prove that the maximum size μ of a matching in G is equal to the maximum value of a flow in N_G. If M is a matching of size μ in G, then f_M has value μ. Hence $\mu\leq v_{max}$. Reciprocally, from Remark 7.12, there is a maximum flow f with integral values. It is in one-to-one correspondence with a matching M of size $v(f)=v_{max}$. Hence, $\mu\geq v_{max}$.

Let $C=(V_s,V_p)$ be a minimum cut. Let $A_s=A\cap V_s$ and $B_s=B\cap V_s$. If there is a vertex $b\in(B\cap N(A_s))\setminus B_s$, then setting $C'=(V_s\cup\{b\},V_p\setminus\{b\})$, we get $\delta(C')\leq\delta(C)-1+1=\delta(C)$. So, by adding the vertices of $(B\cap N(A_s))\setminus B_s$ in V_s if needed, we may assume that the minimum cut that we consider is such that $N(A_s)\subseteq B_s$.

Let us consider the capacity of C.

$$\begin{aligned}\delta_{min}=\delta(C) &= |\{(s,a)\mid a\in A\setminus A_s\}|+|\{(b,p)\mid b\in B_s\}|+|\{(a,b)\}\mid a\in A_s, b\in B\setminus B_s\}|\\ &= |A|-|A_s|+|B_s|\end{aligned}$$

Since $N(A_s)\subseteq B_s$, it follows that $\delta_{min}\geq|A|-|A_s|+|N(A_s)|$. Moreover, the cut $(A_s\cup N(A_s),V(H)\setminus(A_s\cup N(A_s)))$ has capacity $|A|-|A_s|+|N(A_s|$. So $\delta_{min}=|A|-|A_s|+|N(A_s)|$. We conclude that

$$\delta_{min}=\min\{S\subset A\mid |A|-|S|+N(S)\}$$

Hence, from Theorem 7.7, $\mu=v_{max}=\delta_{min}=\min\{S\subset A\mid |A|-|S|+N(S)\}$. □

7.9.3 Maximum-gain closure

In this problem, we have several tasks. Each task $t\in T$ is associated to a gain $g(t)$. The gain may be negative (if so, it corresponds to a loss). We note T^+ (resp. T^-) the set of tasks with positive gain (resp., negative gain).

Besides, there are several *closure* constraints.That is, the choice of a task may imply the choice of one or several other tasks. We represent this closure relation (e.g., implication relation) by an *implication digraph* D_T: its vertices are the tasks and there is an arc (t_1,t_2) if and only if the choice of t_1 implies the choice of t_2.

The objective is to find a set of *compatible*tasks with maximum gain, that is a subset A of T such that:

- there are no arcs leaving A ($E((A,\overline{A}))=\emptyset$) i.e. A is a *closure* in D_T;

- $\sum_{t\in A} g(t)$ is maximum.

Let us show how this problem can be reduced to a problem of cut with minimum capacity, hence to a maximum flow problem. Let $N = (G, s, p, c)$ be the following flow network (See Figure 7.10:

- $V(G) = T \cup \{s, p\}$;
- for any task t with positive gain, link the source s to the task t with an arc (s,t) with capacity $c(s,t) = g(t)$;
- for any task t with non-positive gain, link t to the sink p with an arc (t, p) with capacity $c(t, p) = -g(t)$;
- if a task t_1 implies a task t_2, we add the arc (t_1, t_2) with infinite capacity in the network.

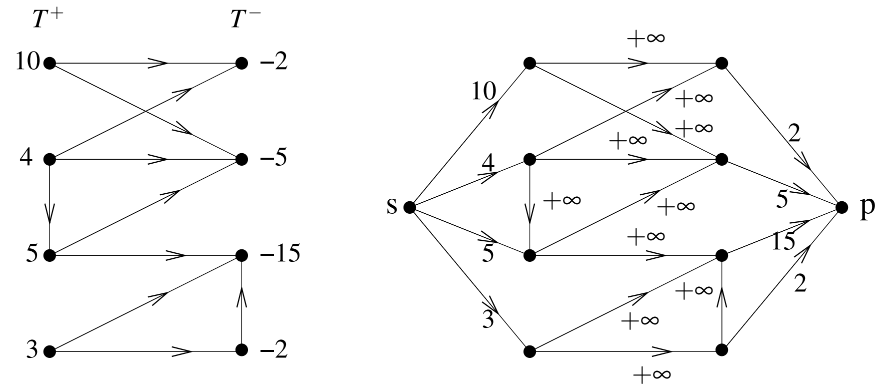


Figure 7.10: An implication digraph and its corresponding flow network

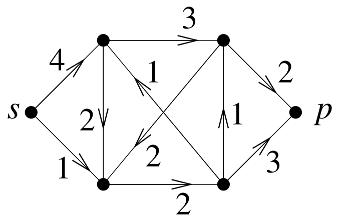
Let us consider a cut $C = (V_s, V_p)$ with finite capacity in the network. We have $S = \{s\} \cup A$ with $A \subset T$. Moreover, since the cut has finite capacity, its arc set contains no arcs of D_T, so A is a set of compatible tasks. Let $A^+ = A \cap T^+$ and $A^- = A \cap T^-$. The capacity of C is:

$$\begin{aligned}
\delta(C) &= \sum_{t \in T^+ \setminus A^+} c(s,t) + \sum_{t \in A^-} c(t,p) \\
&= \sum_{t \in T^+ \setminus A^+} g(t) - \sum_{t \in A^-} g(t) \\
&= \sum_{t \in T^+} g(t) - \sum_{t \in A^+} g(t) - \sum_{t \in A^-} g(t) \\
&= \sum_{t \in T^+} g(t) - \sum_{t \in A} g(t)
\end{aligned}$$

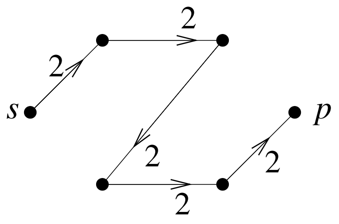
Since $\sum_{t \in T^+} g(t)$, the sum of all positive gains, is a constant, minimizing the capacity of the cut $S = \{s\} \cup A$ is equivalent to maximizing the gain of A.

7.10 Exercices

Exercise 7.1. Let N be the flow network and f_0 the (s,p)-flow in N as depicted in the figure below.



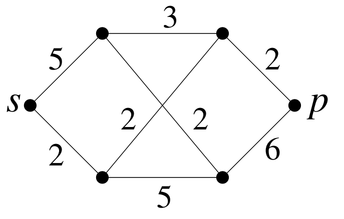
network



initial flow

1) Start from f_0 and find a maximum (s,p)-flow. Detail the steps of the algorithm.
2) Describe a minimum cut of this network.

Exercise 7.2. Find a maximum (s,p)-flow and a minimum (s,p)-cut in the network depicted below. (Detail the steps of the "push" algorithm.)



Exercise 7.3. There are 3 production sites A,B,C and 5 consumption sites $1,2,3,4,5$; their production and consumption, respectively, are given in the following tables.

A	B	C
5	4	7

1	2	3	4	5
3	4	5	2	1

Finally, each production sites can only serve the consumption sites as summarized in the following table.

A	B	C
13	24	345

The problem is to satisfy the consumption sites. Model the following problem in terms of flows and give a solution to the problem or explain why it could not exist.

Exercise 7.4. Prove the following property: *for all (s,p)-cut $C = (V_s,V_p)$, $v(f) = out(f,C) - in(f,C)$*. Why is the hypothesis $s \in V_s$ and $p \in V_p$ important?

Exercise 7.5. Prove Lemma 7.9. Verify that the positivity, the capacity constraint and the flow conservation are satisfied for f'.

Exercise 7.6. The *support* of a flow is the set of arcs on which the flow function is positive. Show that there always exists a maximum flow whose support has no directed cycle.

Exercise 7.7. Let $N = (G, s, p, c)$ be a flow netwrok such that, for all arc e, $c(e)$ is an even integer.

1) Prove that the maximum value of a flow is an even integer. 2) Show that there is a maximum flow f such that, for all arc e, $f(e)$ is an even integer.

Exercise 7.8. Modify Algorithm 7.1 to obtain an algorithm finding a minimum cut in a flow network.

Exercise 7.9. Construct a flow network for which Algorithm 7.1 produces a sequence of flow whose values converges to a finite value when pushing on some sequence of directed paths, while the value of a maximum flow is infinite (or arbitrarily large).

Exercise 7.10. Prove Proposition 7.17.

Exercise 7.11. In this exercise, we study a variant of the algorithm for finding a maximum flow, in which we push the flow along a directed path of maximum residual capacity.

1) Give an algorithm finding a directed path of maximum residual capacity.

2) Show that if we push an amount of x at one step, then we push an amount of at least x at each following step.

Exercise 7.12. Let $N = (G, s, p, c)$ be a flow network. To each vertex $v \in V(G)$, we associate an real $w(v)$. We want to compute a flow f of maximum value satisfying the following extra constraint: $\forall v \in V(G)$ the flow entering v is at most $w(v)$ (i.e. $\sum_{u \in N^-(v)} f(uv) \leq w(v)$). Show how to find such a flow by computing a maximum flow on a network obtained from N by slight modifications.

Exercise 7.13. Deduce Menger's Theorem (5.18) from Theorem 7.7 . (*Hint*: One can use Exercise 7.12 to prove Theorem 5.18-(i).

Exercise 7.14. Deduce Theorem 7.7 from Menger's Theorem (5.18).

Exercise 7.15. Several companies send members to a conference; the ith company send m_i members. During the conference, several workshops are organized simulteanously; the ith workshops can receive at most n_j participants. The organizers want to dispatch participants into workshops so that two members of a same company are not in a same workshop. (The workshop do not need to be full.)

a) Show how to use a flow network for testing if the constraints may be satisfied.

b) If there are p companies and q workshops indexed in such a way that $m_1 \geq \cdots \geq m_p$ and $n_1 \leq \cdots \leq n_q$. Show that there exists a dispatching of participants into groupes satisfying the constraints if and only if, for all $0 \leq k \leq p$ and all $0 \leq l \leq q$, we have $k(q-l) + \sum_{j=1}^{l} n_j \geq \sum_{i=1}^{k} m_i$.

Exercise 7.16 (Unsplittable flow)**.** We consider a flow network with one production site s and many consumption sites, say $p_1, p_2, \dots p_n$. The consumption at site p_i is d_i. We want to route commodities for s to p_i with the following additionnal constraint: the traffic from s to p_i must be routed along a unique directed path (it cannot be split).

Show that this problem is NP-complete.