The maximum flow problem

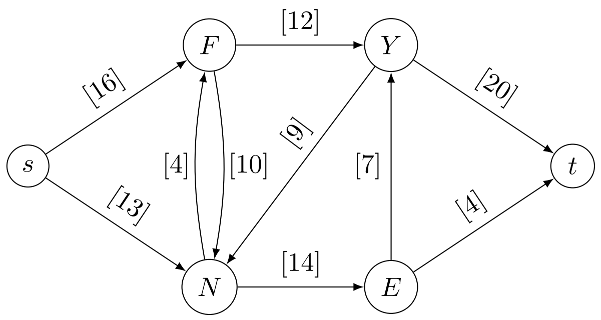
Recherche opérationnelle
Dimitri Watel - ENSIIE

2024

The maximum flow problem is a classic problem in operations research aimed at modeling an optimal movement issue in a graph. The applications are numerous, the most traditional being a fluid network (gas, liquid), from which the problem derives its name, as well as a transportation network, an energy network, or a telecommunications network. We will describe the problem, explain how it can be solved, and demonstrate the correctness of the proposed algorithm.

1 Problem definition

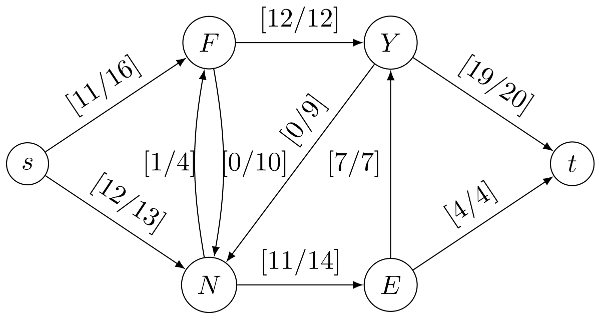
In the following graph G, where a source s and a sink t have been identified, each arc a is associated with an integer $c(a)$. These integers represent the *capacities* of the arcs. We refer to this as a *flow network* or *transport network* (G, s, t, c).

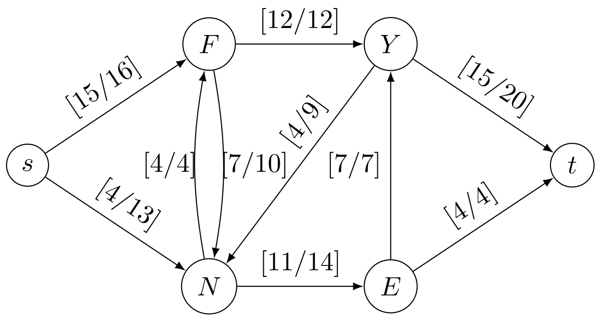


We aim to transport water from s to t by following the edges of the graph, knowing that

- the amount of water entering a node must be equal to the amount exiting the node, except at s and t,
- the amount of water flowing through an edge cannot exceed the capacity of that edge.

In the previous example, the following solutions are feasible.





More formally, a *feasible flow* is a function f from A to $\mathbb{N}$ such that

- for every arc $a \in A$, $f(a) \in [0, c(a)]$ (capacity constraint)
- for every node $v \in V$ **except** s **and** t, $\sum_{a \in \gamma^-(v)} f(a) = \sum_{a \in \gamma^+(v)} f(a)$ (conservation constraint)

Among the two solutions of the example, the first one is more interesting; we observe that the outgoing flow from s is greater, and thus, we were able to send more flow from s to t. We call the *value* the quantity of flow outgoing from s (or equivalently, the quantity incoming to t). Note that there is nothing preventing the flow from looping back and returning to s; in this case, this flow will exit s a second time, but it should not be counted twice in the value. Thus, we define the value v of a flow f with

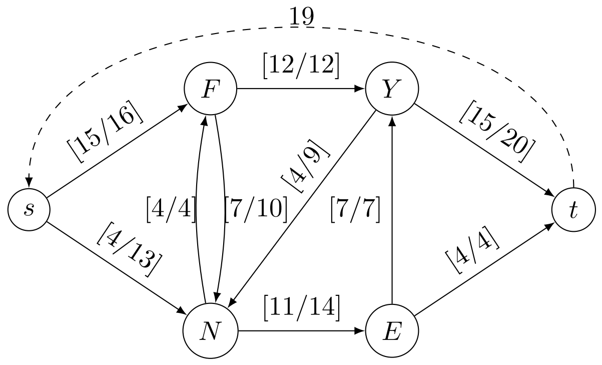
$$v = \sum_{a \in \gamma^-(t)} f(a) - \sum_{a \in \gamma^+(t)} f(a)$$
$$v = \sum_{a \in \gamma^+(s)} f(a) - \sum_{a \in \gamma^-(s)} f(a)$$

The maximum flow problem is described as follows:

Problem 1. Given a flow network (G, s, t, c), find a feasible flow f of maximum value v in this network.

A flow that maximizes v is said to be optimal or maximum.

Before addressing the problem, we can simplify the problem. We notice that the conservation constraint does not involve s and t, but that the flow out of s is equal to the flow into t. By adding a dummy arc between s and t with a flow equal to v, we obtain an feasible flow that satisfies the conservation constraint on all nodes.

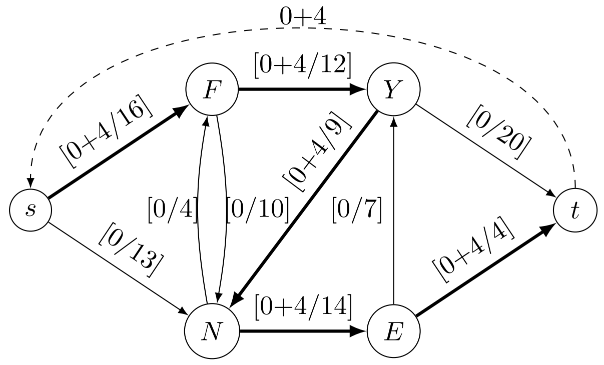


2 The Ford Fulkerson algorithm

2.1 Main idea

The Fold Fulkerson algorithm injects flow into the network gradually until it finds a maximum flow. The algorithm starts with an already feasible flow. The simplest solution is to start from zero flow.

The algorithm searches for *an augmenting path.* This is a path in G connecting s and t along which we can inject flow. A simple example of an augmenting path is a path from s to t where each edge is not saturated (in the sense that the associated flow has not reached its capacity). In the following example, there is an augmenting path with which the flow increases by 4.

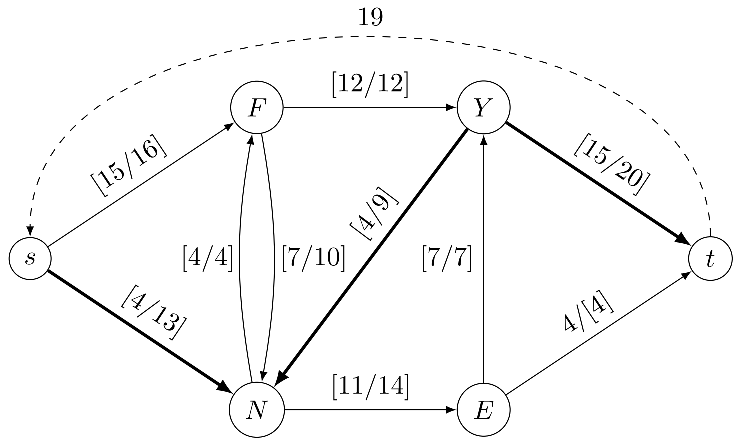


However, this simple version of augmenting paths has its limits. In the example with value 19 at the end of the first section, there is no such path. However, there is a solution with a value of 23 (as indicated in one of the previous examples). The Ford-Fulkerson algorithm is able to overcome this difficulty and find the maximum flow by using a more generalized version of augmenting paths.

Considering a network (G, s, t, c) and a feasible flot f, an *augmenting path* μ is an **(undirected)** path linking s and t such that:

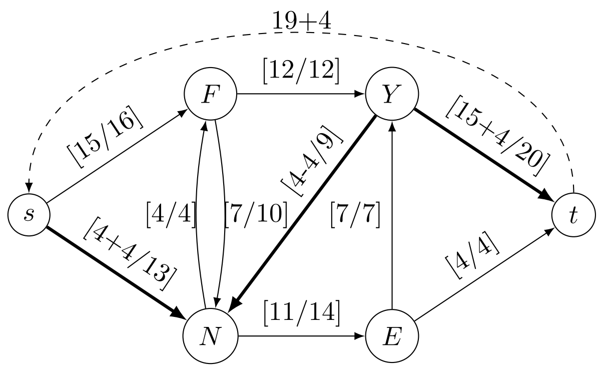
- for every arc a of μ directed from s to t, $f(a) < c(a)$. We write μ^+ those arcs.
- for every arc a of μ directed from t to s, $f(a) > 0$. We write μ^- those arcs.

In the following example, we have represented the chain $sNYt$ (it can be noted that the chain does not take orientation into account). The arcs sN and Yt go from s to t, and these arcs are not saturated. The arc YN goes from t to s, and this arc is not empty. Thus, we indeed have an augmenting chain.



Once we have an augmenting path, we can increase the flow value by increasing $f(a)$ if $a \in \mu^+$ and by decreasing $f(a)$ if $a \in \mu^-$. The change must be the same across all edges, which ensures that we respect the conservation constraint. We also need to respect the capacity constraint, meaning that no edge should exceed its capacity or have a negative flow after the change.

$$v \to v + \min_{a \in \mu} \begin{cases} c(a) - f(a) & \text{if } a \in \mu^+ \\ f(a) & \text{if } a \in \mu^- \end{cases}$$



The Ford Fulkerson algorithm is the following:

Algorithm 1 Ford Fulkerson algorithm

```
Require: $(G, s, t, c)$ a flow network
  $f \leftarrow$ a nul flow
  while il existe une chaîne augmentante $\mu$ dans $G$ dothere exists an augmenting path $\mu$ on $f$
      $dv \leftarrow \min\left(\min_{a\in\mu^+}\{c(a) - f(a)\};\ \min_{a\in\mu^-}\{f(a)\}\right)$
      $\forall a \in \mu^+, f(a) \leftarrow f(a) + dv$
      $\forall a \in \mu^-, f(a) \leftarrow f(a) - dv$
```

2.2 Find an augmenting path

There are two possible methods: the residual network and the marking algorithm.

Let (G, s, t, c) be a flow network and a feasible flow f. We define the *residual network* as a graph $H = (V, B)$ and a weight $\omega : B \to \mathbb{N}$ over the arcs of H such that :

- for every arc $a = (u, v) \in A$ such that $f(a) < c(a)$, we add an arc $b = (u, v) \in B$
- for every arc $a = (u, v) \in A$ such that $f(a) > 0$, we add an arc $b = (v, u) \in B$

Lemma 2.1. *There is an augmenting path in G if and only if there is a path in H from s to t.*

Proof. Let P be a path connecting s and t in G. We denote w_i as the i-th vertex of P, with $s = w_0$.

$$\begin{aligned} & P \text{ is an augmenting path} \\ & \Leftrightarrow \text{for every } i \in [\![0, |P|-1]\!] \\ & \begin{cases} \text{either } (w_i, w_{i+1}) \text{ is an unsaturated arc of } G, \text{ or} \\ (w_{i+1}, w_i) \text{ is a non-empty arc of } G \end{cases} \\ & \Leftrightarrow \text{for every } i \in [\![0, |P|-1]\!], (w_i, w_{i+1}) \text{ is an arc of } H \\ & \Leftrightarrow P \text{ is a path in } H \end{aligned}$$

□

The marking algorithm is the following

Algorithm 2 Marking algorithm

```
Mark the source with +
for $(u, v) \in A$ do
    if $u$ is marked, $v$ is not marked and $f(u, v) < c(u, v)$ then
        mark $v$ with $+(u)$
    else if $v$ is marked, $u$ is not marked and $f(u, v) > 0$, then
        mark $u$ with $-(v)$
Redo Line 2 if at least one node was marked
```

Lemma 2.2. *There is an augmenting path in G if and only if t is marked.*

Proof. Except for s, each marked node is marked with another node from G that has itself been marked in a previous iteration. Thus, by *tracing back* the markings, one always arrives at s. Let us suppose that t is marked and denote $P = (s = w_0, w_1, w_2, \ldots, t = w_{|P|})$ as the sequence of marked nodes that led to the marking of t.

For all $i \in [\![0, |P|-1]\!]$, since w_{i+1} has been marked with either $+w_i$ or $-w_i$,

$$\begin{cases} (w_i, w_{i+1}) \text{ is an unsaturated edge of } G, \text{ or} \\ (w_{i+1}, w_i) \text{ is a non-empty edge of } G \end{cases}$$

Therefore, P is an augmenting path.

If $P = (s = w_0, w_1, w_2, \ldots, t = w_{|P|})$ is an augmenting path. We will show by induction that each node in P is marked. w_0 is marked in the first line of the algorithm. Now suppose that w_i is marked by the algorithm during some iteration. We focus on the execution of the loop in line 2 that follows this marking. If, during this execution, w_{i+1} is already marked (by some node other than w_i), then we have proven the desired induction property. Otherwise, since P is an augmenting path,

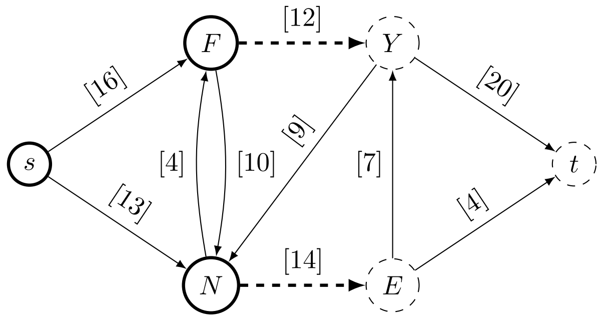
$$\begin{cases} (w_i, w_{i+1}) \text{ is an unsaturated edge of } G, \text{ or} \\ (w_{i+1}, w_i) \text{ is a non-empty edge of } G \end{cases}$$

Depending on the case, it is possible to mark w_{i+1} with either $+w_i$ or $-w_i$. Thus, w_{i+1} will necessarily be marked no later than during the iteration of the loop in line 2 where u and v are the nodes w_i and w_{i+1}. By induction, all nodes in P are marked, particularly t. □

The last section proves that the Ford-Fulkerson algorithm is correct by showing that, if there are no more augmenting paths, then the flow is maximal.

3 The min-cut problem and the strong duality theorem

We will focus on a related problem, a dual problem of the maximum flow problem, to demonstrate the accuracy of the algorithm. We are interested in cuts of the graph that separate s and t. Such a cut is a partition of the nodes into two sets (S, T) such that $s \in S$ and $t \in T$. Below is an example of a cut where the nodes in bold correspond to S and the others to T. The two sets do not need to be of the same size and may contain only s or only t.



Given a cut, we are interested in the arcs going from S to T (and only those). We then calculate the sum of the capacities of these arcs. In the previous example, these arcs are represented as dashed lines. For instance, we note that (Y, N) is not counted because this arc goes from T to S. The capacity of this cut is therefore 26.

Remark 1. "Even if the dummy arc is added to the network, it is never counted in the capacity of the cut because it necessarily connects T to S and not the other way around.

The minimum cut problem is described as follows:

Problem 2. Given a flow network (G, s, t, c), find a cut (S, T) of minimum capacity in this network.

Such a cut is called a minimum cut.

Although different from the maximum flow problem, we will demonstrate that the two problems are related: we will show that the value of the optimal solutions in the same flow network is the same in both problems. If we know the value of a maximum flow, then we know the capacity of a minimum cut and vice versa. To prove this theorem, we will first demonstrate an intermediate result.

Lemma 3.1. *Given a flow network $(G = (V, A), s, t, c)$ with the dummy arc (t, s) and $W \subset V$ then $\sum_{a\in\gamma^-(W)} f(a) = \sum_{a\in\gamma^+(W)} f(a)$.*

Remark 2. In other words, the conservation constraint is satisfied on every subset of nodes.

Proof. Since the dummy arc is present, the conservation constraint is satisfied by every node of W.

$$\sum_{a\in\gamma^-(v)} f(a) = \sum_{a\in\gamma^+(v)} f(a) \;\forall v \in W$$

$$\sum_{v\in W}\sum_{a\in\gamma^-(v)} f(a) = \sum_{v\in W}\sum_{a\in\gamma^+(v)} f(a)$$

Let's consider an arc $a = (v_1, v_2) \in A$. If v_1 and v_2 are in W, then this arc appears in both the left sum and the right sum. If we simplify, on the left side only the arcs (u, v) such that $u \notin W$ remain, and on the right side only the arcs (v, u) such that $u \notin W$ remain.

$$\sum_{v\in W}\sum_{\substack{(u,v)\in\gamma^-(v)\\ u\notin W}} f(u,v) = \sum_{v\in W}\sum_{\substack{(v,u)\in\gamma^+(v)\\ u\notin W}} f(v,u)$$

$$\sum_{a\in\gamma^-(W)} f(a) = \sum_{a\in\gamma^+(W)} f(a)$$

□

We can now demonstrate a weak version of the theorem we would like to prove.

Theorem 3.1. *Given a flow network $(G = (V, A), s, t, c)$, an admissible flow f of value v, and a cut (S, T) of capacity $c(S, T)$, then $v \leq c(S, T)$.*

Proof. By Lemma 3.1,

$$\sum_{a\in\gamma^-(S)} f(a) = \sum_{a\in\gamma^+(S)} f(a)$$

One of the arcs entering S is the dummy arc (t, s), with flow v. Since all the other arcs in $\gamma^-(S)$ have positive flow, we have

$$v \leq \sum_{a\in\gamma^+(S)} f(a)$$

Since every arc has a flow lower than its capacity

$$v \leq \sum_{a\in\gamma^+(S)} c(a)$$

Finally, since every output arc of S goes toward T, the last sum is the capacity of the cut.

$$v \leq c(S, T)$$

□

If, given a flow, we find a cut whose capacity is equal to the value of the flow, then this flow is necessarily maximum (and the cut is minimum). We will show that this is always possible and that, starting from the flow resulting from the Ford Fulkerson algorithm, we can construct this cut quite simply.

Theorem 3.2. *Given a flow network $(G = (V, A), s, t, c)$, the value v of a maximum flow, and the capacity C of a minimul cut, then $v = C$.*

Theorem 3.3. *The Ford Fulkerson algorithm returns a maximum flow.*

Proof. The proof of the two theorems is done jointly. Let us consider the flow f of value v returned at the end of the algorithm. By construction, the flow network with f no longer has an augmenting path. According to Lemma 2.2, if we apply the labeling algorithm, the node t is not labeled. Since the node s is still labeled, we obtain a cut (S, T) by putting all labeled nodes in S and the others in T.

By Lemma 3.1,

$$\sum_{a\in\gamma^-(S)} f(a) = \sum_{a\in\gamma^+(S)} f(a)$$

One of the arcs entering S is the dummy arc (t, s), with flow v. All the other arcs in $\gamma^-(S)$ are arcs (u, v) where $u \in T$ and $v \in S$. Since v is marked and u is not, it follows that $f(u, v) = 0$ (otherwise, u would have been marked with "$-v$").

$$v = \sum_{a\in\gamma^+(S)} f(a)$$

All the arcs in $\gamma^+(S)$ are arcs (u, v) where $u \in S$ and $v \in T$. Since u is marked and v is not, it follows that $f(u, v) = c(u, v)$ (otherwise, v would have been marked with "$+u$").

$$v = \sum_{a\in\gamma^+(S)} c(a)$$

Finally, since every output arc of S goes toward T, the last sum is the capacity of the cut.

$$v = c(S, T)$$

According to theorem 3.1, any flow has a value less than $c(S, T)$, therefore less than v. Similarly, every cut has a capacity greater than v which is greater than $c(S, T)$. We can thus conclude that the flow f returned by the Ford Fulkerson algorithm is a maximum flow and that (S, T) is a minimum cut. $\square$

To conclude this section, let us just focus on the method described in the proof of Theorem 3.3 to find a minimum cut:

Algorithm 3 Find a minimum cut

```
Run the Ford Fulkerson algorithm
# Version 1
After the last iteration, use the marking algorithm
$S \leftarrow$ the marked nodes
$T \leftarrow$ the other nodes
return $(S, T)$
# Version 2
After the last iteration, build the residual network $H$
$S \leftarrow$ the nodes that can be accessed from $s$ in $H$
$T \leftarrow$ the other nodes
return $(S, T)$
```