Maximum flow problem and Ford-Fulkerson Algorithm

Ford–Fulkerson algorithm

The Ford–Fulkerson method (named for L. R. Ford, Jr. and D. R. Fulkerson) is an algorithm which computes the maximum flow in a flow network. It was published in 1956. The name "Ford–Fulkerson" is often also used for the Edmonds–Karp algorithm, which is a specialization of Ford–Fulkerson.

The idea behind the algorithm is simple. As long as there is a path from the source (start node) to the sink (end node), with available capacity on all edges in the path, we send flow along one of these paths. Then we find another path, and so on. A path with available capacity is called an **augmenting path**.

The Problem

- Consider, for example, a highway system in which the edges are highways and the nodes are interchanges
- A computer network in which the edges are links that can carry packets and the nodes are Switches
- A fluid network in which edges are pipes that carry liquid, and the nodes are junctures where pipes are plugged together.

Network models of this type have several ingredients:

- ***Capacities*** on the edges, indicating how much they can carry;
- ***Source*** nodes in the graph, which generate traffic;
- ***Sink*** (or destination) nodes in the graph, which can "absorb" traffic as it arrives;

Flow Networks

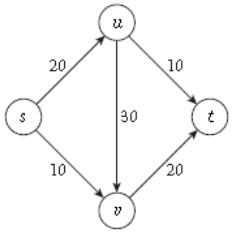
Traffic as flow—an abstract entity that is generated at source nodes, transmitted across edges, and absorbed at sink nodes. Formally, we'll say that a flow network is a directed graph G = (V, E) with the following features:

- Associated with each edge e is a **capacity**, which is a nonnegative number that we denote $\mathbf{c_e}$.
- There is a single *source* node $s \in V$.
- There is a single *sink* node $t \in V$.
- Nodes other than *s* and *t* will be called ***internal* nodes**.

Two **assumptions** about the flow networks are:

- No edge enters the source *s* and no edge leaves the sink *t*
- There is at least one edge incident to each node
- All capacities are integers.

The following figure illustrates a flow network with four nodes and five edges, and capacity values given next to each edge.



Defining Flow

We say that an *s-t flow* is a function *f* that maps each edge *e* to a nonnegative real number, $f : E \rightarrow \mathbf{R}^+$; the value *f (e)* intuitively represents the amount of flow carried by edge *e*. A flow *f* must satisfy the following two properties:

(i) *(Capacity conditions)* For each $e \in E$, we have $0 \leq f(e) \leq c_e$.
(ii) *(Conservation conditions)* For each node *v* other than *s* and *t*, we have

$$\sum_{e \text{ into } v} f(e) = \sum_{e \text{ out of } v} f(e).$$

- Here $\sum_{e \text{ into } v} f(e)$ sums the flow value *f (e)* over all edges entering node .
- $\sum_{e \text{ out of } v} f(e)$ is the sum of flow values over all edges leaving node *v*.

For every node other than the source and the sink, the amount of flow entering must equal the amount of flow leaving.

The *value* of a flow *f* , denoted *v(f)*, is defined to be the amount of flow generated at the source:

$$v(f) = \sum_{e \text{ out of } s} f(e).$$

To make the notation more compact, we define

$$f^{\text{out}}(v) = \sum_{e \text{ out of } v} f(e)$$

$$f^{\text{in}}(v) = \sum_{e \text{ into } v} f(e).$$

The Maximum-Flow Problem Given a flow network, a natural goal is to arrange the traffic so as to make as efficient use as possible of the available capacity. Thus the basic algorithmic problem will consider the following: Given a flow network, find a flow of maximum possible value.

Designing the Algorithm

The more general way of pushing flow: We can push ***forward*** on edges with **leftover capacity**, and we can push ***backward*** on edges that are **already carrying flow**, to divert it in a different direction.

The Residual Graph Given a flow network *G*, and a flow *f* on *G*, we define the *residual graph* G_f of *G* with respect to *f* as follows.

- The node set of G_f is the same as that of *G*.

- For each edge $e = (u, v)$ of G on which $f(e) < c_e$, there are $c_e - f(e)$ "leftover" units of capacity on which we could try pushing flow **forward**. So we include the edge $e = (u, v)$ in G_f, with a capacity of $c_e - f(e)$. We will call edges included this way ***forward edges***.
- For each edge $e = (u, v)$ of G on which $f(e) > 0$, there are $f(e)$ units of flow that we can "undo" if we want to, by pushing flow backward. So we include the edge $e' = (v, u)$ in G_f, with a capacity of $f(e)$.
    - Note that e' has the same ends as e, but its direction is reversed; we will call edges included this way ***backward edges***.

```
augment(f, P)
  Let b = bottleneck(P).
  For each edge e ∈ P
    If e is a forward edge then
      increase f(e) in G by b.
    Else (e is a backward edge)
      decrease f(e) in G by b.
    Endif
  Endfor
  Return(f)
```

Let P be a simple s-t path in G_f— that is, P does not visit any node more than once.

We define bottleneck(P, f) to be the minimum residual capacity of any edge on P, with respect to the flow f.

```
Max-Flow (G, s, t, c)
  Initially f(e) = 0 for all e in G.
  While there is an s-t path in the residual graph G_f
    Let P be a simple s-t path in G_f
    f' = augment(f, P)
    Update f to be f'
    Update the residual graph G_f to be G_f'
  Endwhile
  Return f
```