Wikipedia
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Languages
Català
Čeština
Deutsch
Español
فارسی
한국어
Հայերեն
עברית
日本語
Polski
Русский
Tagalog
ไทย
Українська
Tiếng Việt
中文
Edit links

# Flow network

From Wikipedia, the free encyclopedia

In graph theory, a **flow network** (also known as a **transportation network**) is a directed graph where each edge has a **capacity** and each edge receives a flow. The amount of flow on an edge cannot exceed the capacity of the edge. Often in operations research, a directed graph is called a **network**. The vertices are called **nodes** and the edges are called **arcs**. A flow must satisfy the restriction that the amount of flow into a node equals the amount of flow out of it, unless it is a **source**, which has only outgoing flow, or **sink**, which has only incoming flow. A network can be used to model traffic in a road system, circulation with demands, fluids in pipes, currents in an electrical circuit, or anything similar in which something travels through a network of nodes.

**Contents** [hide]

1 Definition
2 Example
3 Applications
4 See also
5 References
6 Further reading
7 External links

## Definition [edit]

Let $G = (V, E)$ be a finite directed graph in which every edge $(u, v) \in E$ has a non-negative, real-valued capacity $c(u, v)$. If $(u, v) \notin E$, we assume that $c(u, v) = 0$. We distinguish two vertices: a source $s$ and a sink $t$. A flow in a flow network is a real function $f : V \times V \to \mathbb{R}$ with the following three properties for all nodes $u$ and $v$:

| | |
|---|---|
| **Capacity constraints**: | $f(u, v) \leq c(u, v)$. The flow along an edge cannot exceed its capacity. |
| **Skew symmetry**: | $f(u, v) = -f(v, u)$. The net flow from $u$ to $v$ must be the opposite of the net flow from $v$ to $u$ (see example). |
| **Flow conservation**: | $\sum_{w \in V} f(u, w) = 0$, unless $u = s$ or $u = t$. The net flow to a node is zero, except for the source, which "produces" flow, and the sink, which "consumes" flow. |

i.e. **Flow conservation** implies: $\sum_{(u,v)\in E} f(u, v) = \sum_{(v,z)\in E} f(v, z)$, for each vertex $v \in V \setminus \{s, t\}$

Notice that $f(u, v)$ is the *net* flow from $u$ to $v$. If the graph represents a physical network, and if there is a real flow of, for example, 4 units from $u$ to $v$, and a real flow of 3 units from $v$ to $u$, we have $f(u, v) = 1$ and $f(v, u) = -1$.

Basically we can say that flow for a physical network is flow leaving at s = $\sum_{(s,v)\in E} f(s, v)$

The **residual capacity** of an edge is $c_f(u, v) = c(u, v) - f(u, v)$. This defines a **residual network** denoted $G_f(V, E_f)$, giving the amount of *available* capacity. See that there can be a path from $u$ to $v$ in the residual network, even though there is no path from $u$ to $v$ in the original network. Since flows in opposite directions cancel out, *decreasing* the flow from $v$ to $u$ is the same as *increasing* the flow from $u$ to $v$. An **augmenting path** is a path $(u_1, u_2, \dots, u_k)$ in the residual network, where $u_1 = s$, $u_k = t$, and $c_f(u_i, u_{i+1}) > 0$. A network is at maximum flow if and only if there is no augmenting path in the residual network $G_f$ .

So $G_f$ is constructed using graph G as follows:

1. Vertices of $G_f$ = $V$

2. Edges of $G_f$ = $E_f$ defined as-

For each edge $(x,y) \in E$

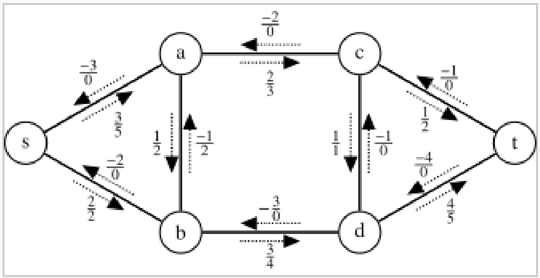
```
    (i). If $f(x,y) < c(x,y)$, make Forward edge $(x,y) \in E_f$ with **capacity**
$c_f = c(x,y) - f(x,y)$.
    (ii). If $f(x,y) > 0$, make Backward edge $(y,x) \in E_f$ with **capacity**
$c_f = f(x,y)$.
```

This concept is used in Ford–Fulkerson algorithm which computes the maximum flow in a flow network.

Sometimes one needs to model a network with more than one source, a **supersource** is introduced to the graph.[1] This consists of a vertex connected to each of the sources with edges of infinite capacity, so as to act as a global source. A similar construct for sinks is called a **supersink**.[2]

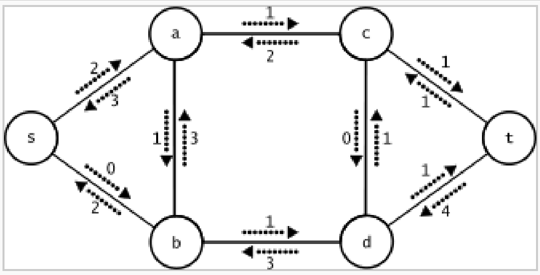
## Example [edit]

To the right you see a flow network with source labeled $s$, sink $t$, and four additional nodes. The flow and capacity is denoted $f/c$. Notice how the network upholds skew symmetry, capacity constraints and flow conservation. The total amount of flow from $s$ to $t$ is 5, which can be easily seen from the fact that the total outgoing flow from $s$ is 5, which is also the incoming flow to $t$. We know that no flow appears or disappears in any of the other nodes.



A flow network showing flow and capacity

Below you see the residual network for the given flow. Notice how there is positive residual capacity on some edges where the original capacity is zero, for example for the edge $(d,c)$. This flow is not a maximum flow. There is available capacity along the paths $(s,a,c,t)$, $(s,a,b,d,t)$ and $(s,a,b,d,c,t)$, which are then the augmenting paths. The residual capacity of the first path is



Residual network for the above flow network, showing residual capacities.

$\min(c(s,a)-f(s,a), c(a,c)-f(a,c), c(c,t)-f(c,t))$
$= \min(5-3, 3-2, 2-1) = \min(2,1,1) = 1$. Notice that as long as there exists some path with a positive residual capacity, the flow will not be maximum. The residual capacity for some path is the minimum residual capacity of all edges in that path.

## Applications [edit]

*See also: Pipe network analysis*

Picture a series of water pipes, fitting into a network. Each pipe is of a certain diameter, so it can only maintain a flow of a certain amount of water. Anywhere that pipes meet, the total amount of water coming into that junction must be equal to the amount going out, otherwise we would quickly run out of water, or we would have a buildup of water. We have a water inlet, which is the source, and an outlet, the sink. A flow would then be one possible way for water to get from source to sink so that the total amount of water coming out of the outlet is consistent. Intuitively, the total flow of a network is the rate at which water comes out of the outlet.

Flows can pertain to people or material over transportation networks, or to electricity over electrical distribution systems. For any such physical network, the flow coming into any intermediate node needs to equal the flow going out of that node. This conservation constraint was formalized as Kirchhoff's current law.

Flow networks also find applications in ecology: flow networks arise naturally when considering the flow of nutrients and energy between different organizations in a food web. The mathematical problems associated with such networks are quite different from those that arise in networks of fluid or traffic flow. The field of ecosystem network analysis, developed by Robert Ulanowicz and others, involves using concepts from information theory and thermodynamics to study the evolution of these networks over time.

The simplest and most common problem using flow networks is to find what is called the maximum flow, which provides the largest possible total flow from the source to the sink in a given graph. There are many other problems which can be solved using max flow algorithms, if they are appropriately modeled as flow networks, such as bipartite matching, the assignment problem and the transportation problem. Maximum flow problems can be solved efficiently with the relabel-to-front algorithm. The max-flow min-cut theorem states that finding a maximal network flow is equivalent to finding a cut of minimum capacity that separates the source and the sink. Where a cut is the division of vertices such that the source is in one division and the sink is in another.

In a multi-commodity flow problem, you have multiple sources and sinks, and various "commodities" which are to flow from a given source to a given sink. This could be for example various goods that are produced at various factories, and are to be delivered to various given customers through the *same* transportation network.

In a minimum cost flow problem, each edge $u, v$ has a given cost $k(u, v)$, and the cost of sending the flow $f(u, v)$ across the edge is $f(u, v) \cdot k(u, v)$. The objective is to send a given amount of flow from the source to the sink, at the lowest possible price.

In a circulation problem, you have a lower bound $l(u, v)$ on the edges, in addition to the upper bound $c(u, v)$. Each edge also has a cost. Often, flow conservation holds for *all* nodes in a circulation problem, and there is a connection from the sink back to the source. In this way, you can dictate the total flow with $l(t, s)$ and $c(t, s)$. The flow *circulates* through the network, hence the name of the problem.

In a **network with gains** or **generalized network** each edge has a **gain**, a real number (not zero) such that, if the edge has gain *g*, and an amount *x* flows into the edge at its tail, then an amount *gx* flows out at the head.

In a **source localization problem**, an algorithm tries to identify the most likely source node of information diffusion through a partially observed network. This can be done in linear time for trees and cubic time for arbitrary networks and has applications ranging from tracking mobile phone users to identifying the originating village of disease outbreaks.[3]

## See also [edit]

- Braess' paradox
- Centrality
- Constructal theory
- Ford–Fulkerson algorithm
- Dinic's algorithm
- Flow (computer networking)
- Flow graph
- Max-flow min-cut theorem
- Oriented matroid
- Shortest path problem

## References [edit]

1. ^ Black, Paul E. "Supersource". *Dictionary of Algorithms and Data Structures*. NIST.
2. ^ Black, Paul E. "Supersink". *Dictionary of Algorithms and Data Structures*. NIST.
3. ^ http://www.pedropinto.org.s3.amazonaws.com/publications/locating_source_diffusion_networks.pdf

## Further reading [edit]

- George T. Heineman, Gary Pollice, and Stanley Selkow (2008). "Chapter 8:Network Flow Algorithms". *Algorithms in a Nutshell*. Oreilly Media. pp. 226–250. ISBN 978-0-596-51624-6.
- Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin (1993). *Network Flows: Theory, Algorithms and Applications*. Prentice Hall. ISBN 0-13-617549-X.
- Bollobás, Béla (1979). *Graph Theory: An Introductory Course*. Heidelberg: Springer-Verlag. ISBN 3-540-90399-2.
- Chartrand, Gary & Oellermann, Ortrud R. (1993). *Applied and Algorithmic Graph Theory*. New York: McGraw-Hill. ISBN 0-07-557101-3.
- Even, Shimon (1979). *Graph Algorithms*. Rockville, Maryland: Computer Science Press. ISBN 0-914894-21-8.
- Gibbons, Alan (1985). *Algorithmic Graph Theory*. Cambridge: Cambridge University Press. ISBN 0-521-28881-9.
- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein (2001) [1990]. "26".

*Introduction to Algorithms* (2nd ed.). MIT Press and McGraw-Hill. pp. 696–697. ISBN 0-262-03293-7.

## External links [edit]

- Maximum Flow Problem
- Real graph instances
- Lemon C++ library with several maximum flow and minimum cost circulation algorithms
- QuickGraph, graph data structures and algorithms for .Net

Categories: Network flow | Graph algorithms | Operations research | Directed graphs