Flow network

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 computer network, circulation with demands, fluids in pipes, currents in an electrical circuit, or anything similar in which something travels through a network of nodes.

Contents

Definition

A **network** is a graph $G = (V, E)$, where V is a set of vertices and E is a set of V's edges – a subset of $V \times V$ – together with a non-negative function $c\colon\ V \times V \to \mathbb{R}_\infty$, called the **capacity** function. Without loss of generality, we may assume that if $(u, v)\ \therefore\ E$ then (v, u) is also a member of E, since if $(v, u)\ \varnothing\ E$ then we may add (v, u) to E and then set $c(v, u) = 0$.

If two nodes in G are distinguished, a source s and a sink t, then (G, c, s, t) is called a **flow network**.[1]

Flows

There are various notions of a flow function that can be defined in a flow graph. Flow functions model the net flow of units between pairs of nodes, and are useful when asking questions such as *what is the maximum number of units that can be transferred from the source node s to the sink node t?* The simplest example of a flow function is known as a pseudo-flow.

> A **pseudo-flow** is a function $f\colon\ V \times V \to \mathbb{R}$ that satisfies the following two constraints for all nodes u and v:

- *Skew symmetry*: Only encode the net flow of units between a pair of nodes u and v (see intuition below), that is: $f(u, v) = -f(v, u)$.
- *Capacity constraint*: An arc's flow cannot exceed its capacity, that is: $f(u, v) \leq c(u, v)$.

Given a pseudo-flow f in a flow network, it is often useful to consider the net flow entering a given node v, that is, the sum of the flows entering v. The **excess** function $x_f\colon V \to \mathbb{R}$ is defined by $x_f(u) = \sum_{v\,\therefore\,V} f(v, u)$. A node u is said to be **active** if $x_f(u) > 0$, **deficient** if $x_f(u) < 0$ or **conserving** if $x_f(u) = 0$.

These final definitions lead to two strengthenings of the definition of a pseudo-flow:

A **pre-flow** is a pseudo-flow that, for all $v \therefore V\setminus\{s\}$, satisfies the additional constraint:

- *Non-deficient flows*: The net flow *entering* the node v is non-negative, except for the source, which "produces" flow. That is: $x_f(v) \geq 0$ for all $v \therefore V\setminus\{s\}$.

A **feasible flow**, or just a **flow**, is a pseudo-flow that, for all $v \therefore V\setminus\{s, t\}$, satisfies the additional constraint:

- *Flow conservation*: The net flow *entering* the node v is 0, except for the source, which "produces" flow, and the sink, which "consumes" flow. That is: $x_f(v) = 0$ for all $v \therefore V\setminus\{s, t\}$.

The **value** of a feasible flow f, denoted $|f|$, is the net flow into the sink t of the flow network. That is, $|f| = x_f(t)$.

Intuition

In the context of flow analysis, there is only an interest in considering how units are transferred between nodes in a holistic sense. Put another way, it is not necessary to distinguish multiple arcs between a pair of nodes:

- Given any two nodes u and v, if there are two arcs from u to v with capacities 5 and 3 respectively, this is equivalent to considering only a single arc between u and v with capacity 8 — it is only useful to know that 8 units can be transferred from u to v, not how they can be transferred.
- Again, given two nodes u and v, if there is a flow of 5 units from u to v, and another flow of 3 units from v to u, this is equivalent to a net flow of 2 units from u to v, or a net flow of -2 units from v to u (so sign indicates direction) — it is only useful to know that a net flow of 2 units will flow between u and v, and the direction that they will flow, not how that net flow is achieved.

For this reason, the *capacity function* $c\colon V \times V \to \mathbb{R}_\infty$, which does not allow for multiple arcs starting and ending at the same nodes, is sufficient for flow analysis. Similarly, it is enough to impose the *skew symmetry* constraint on flow functions to ensure that flow between two vertices is encoded by a single number (to indicate magnitude), and a sign (to indicate direction) — by knowing the flow between u and v you implicitly, via skew symmetry, know the flow between v and u. These simplifications of the model aren't always immediately intuitive, but they are convenient when it comes time to analyze flows.

The *capacity constraint* simply ensures that a flow on any one arc within the network cannot exceed the capacity of that arc.

Concepts useful to flow problems

Residuals

The **residual capacity** of an arc with respect to a pseudo-flow f, denoted c_f, is the difference between the arc's capacity and its flow. That is, $c_f(e) = c(e) - f(e)$. From this we can construct a **residual network**, denoted $G_f(V, E_f)$, which models the amount of *available* capacity on the set of arcs in $G = (V, E)$. More formally, given a flow network

G, the residual network G_f has the node set V, arc set $E_f = \{e \therefore V \times V : c_f(e) > 0\}$ and capacity function c_f.

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

Note 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.

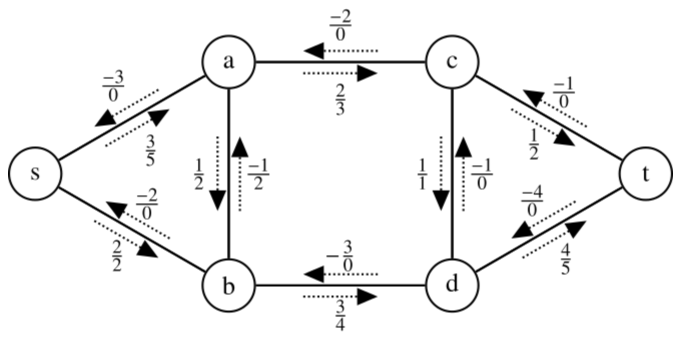
Augmenting paths

An **augmenting path** is a path $(u_1, u_2, \ldots, u_k)$ 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.

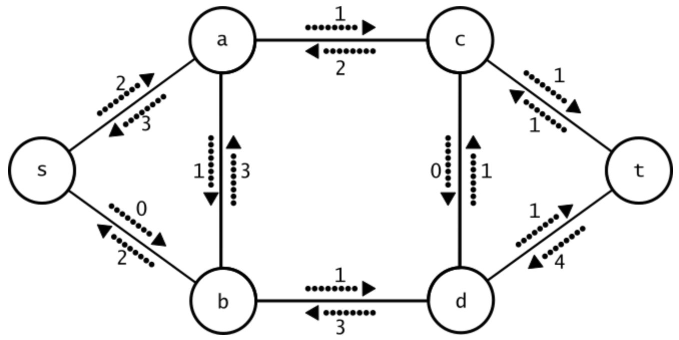
Multiple sources and/or sinks

Sometimes, when modeling a network with more than one source, a **supersource** is introduced to the graph.[2] 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**.[3]

Example



A flow network showing flow and capacity



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

To the left 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.

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

$\min(c(s,a) - f(s,a), c(a,c) - f(a,c), c(c,t) - f(c,t)) = \min(5 - 3, 3 - 2, 2 - 1) = \min(2, 1, 1) = 1$. 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

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 is equivalent to 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 organisms 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.

Classifying flow problems

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.

Well-known algorithms for the Maximum Flow Problem

Inventor(s)	Year	Time complexity (with n nodes and m arcs)
Edmonds–Karp algorithm	1972	$O(m^2n)$
MPM (Malhotra, Pramodh-Kumar and Maheshwari) algorithm[4]	1978	$O(n^3)$
James B. Orlin[5]	2013	$O(mn)$

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.[6]

See also

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

References

1. A.V. Goldberg, É. Tardos and R.E. Tarjan, Network flow algorithms, Tech. Report STAN-CS-89-1252, Stanford University CS Dept., 1989
2. This article incorporates public domain material from the NIST document: Black, Paul E. "Supersource" (https://xlinux.nist.gov/dads/HTML/supersource.html). *Dictionary of Algorithms and Data Structures*.
3. This article incorporates public domain material from the NIST document: Black, Paul E. "Supersink" (https://xlinux.nist.gov/dads/HTML/supersink.html). *Dictionary of Algorithms and Data Structures*.
4. Malhotra, V.M.; Kumar, M.Pramodh; Maheshwari, S.N. (1978). "An $O(|V|^3)$ algorithm for finding maximum flows in networks". *Information Processing Letters*. **7** (6): 277–278. doi:10.1016/0020-0190(78)90016-9 (https://doi.org/10.1016%2F0020-0190%2878%2990016-9).
5. Orlin, J. B. (2013). "Max flows in O(nm) time, or better" (http://jorlin.scripts.mit.edu/docs/publications/O(nm)MaxFlow.pdf) (PDF). *Proceedings of the 2013 Symposium on the Theory of Computing*: 765–774. Archived at
6. http://www.pedropinto.org.s3.amazonaws.com/publications/locating_source_diffusion_networks.pdf

Further reading

- George T. Heineman; Gary Pollice; 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

- Maximum Flow Problem (http://www-b2.is.tokushima-u.ac.jp/~ikeda/suuri/maxflow/Maxflow.shtml)
- Real graph instances (http://www.dis.uniroma1.it/~challenge9/download.shtml)
- Lemon C++ library with several maximum flow and minimum cost circulation algorithms (http://lemon.cs.elte.hu/)
- QuickGraph (http://quickgraph.codeplex.com/), graph data structures and algorithms for .Net