Flow networks

**Definition.** A *flow network* is a directed graph \( G = (V, E) \) with two distinguished vertices: a *source* \( s \) and a *sink* \( t \). Each edge \((u, v) \in E\) has a nonnegative *capacity* \( c(u, v) \). If \((u, v) \notin E\), then \( c(u, v) = 0 \).

**Example:**

![Flow network example](image)

### Flow networks

**Definition.** A *positive flow* on \( G \) is a function \( p : V \times V \to \mathbb{R} \) satisfying the following:

- **Capacity constraint:** For all \( u, v \in V \),
  \[ 0 \leq p(u, v) \leq c(u, v). \]
- **Flow conservation:** For all \( u \in V - \{s, t\} \),
  \[ \sum_{v \in V} p(u, v) - \sum_{v \in V} p(v, u) = 0. \]

The *value* of a flow is the net flow out of the source:

\[ \sum_{v \in V} p(s, v) - \sum_{v \in V} p(v, s). \]

### A flow on a network

![Flow example](image)

**Flow conservation**

- Flow into \( u \) is \( 2 + 1 = 3 \).
- Flow out of \( u \) is \( 0 + 1 + 2 = 3 \).

The value of this flow is \( 1 - 0 + 2 = 3 \).

### The maximum-flow problem

**Maximum-flow problem:** Given a flow network \( G \), find a flow of maximum value on \( G \).

![Maximum-flow example](image)

The value of the maximum flow is \( 4 \).

### Application: Bipartite Matching

A graph \( G(V,E) \) is called *bipartite* if \( V \) can be partitioned into two sets \( V = A \cup B \), and each edge of \( E \) connects a vertex of \( A \) to a vertex of \( B \).

A *matching* is a set of edges \( M \) of \( E \), where each vertex of \( A \) is adjacent to at most one vertex of \( B \).
Matching and flow problem

Add a vertex \( s \), and connect it to each vertex of \( A \).
Add a vertex \( t \), and connect each vertex of \( B \) to \( t \).
The capacity of all edges is 1.

Find max flow. Assume it is an integer flow, so the flow of each edge is either 0 or 1.

Each edge of \( G \) that carries flow is in the matching.
Each edge of \( G \) that does not carry flow is not in the matching.

**Claim:** The edge between \( A \) and \( B \) that carry flow form a matching.

Flow cancellation

Without loss of generality, positive flow goes either from \( u \) to \( v \), or from \( v \) to \( u \), but not both.

Net flow from \( u \) to \( v \) in both cases is 1.

The capacity constraint and flow conservation are preserved by this transformation.

Equivalence of definitions

Net flow vs. positive flow.

**Theorem.** The two definitions are equivalent.

**Proof.** (\( \Rightarrow \)) Let \( f(u, v) = p(u, v) - p(v, u) \).

- **Capacity constraint:** Since \( p(u, v) \leq c(u, v) \) and \( p(v, u) \geq 0 \), we have \( f(u, v) \leq c(u, v) \).

- **Flow conservation:**
  \[
  \sum_{v \in V} f(u, v) = \sum_{v \in V} (p(u, v) - p(v, u))
  = \sum_{v \in V} p(u, v) - \sum_{v \in V} p(v, u)
  \]

- **Skew symmetry:**
  If \( u \in V - \{s, t\} \), then \( \sum_{v \in V} f(u, v) = 0 \).

   \[
   f(u, v) = p(u, v) - p(v, u)
   = -(p(v, u) - p(u, v))
   = -f(v, u).
   \]

A notational simplification

**Idea:** Work with the net flow between two vertices, rather than with the positive flow.

**Definition.** A (net) flow on \( G \) is a function \( f : V \times V \to \mathbb{R} \) satisfying the following:

- **Capacity constraint:** For all \( u, v \in V \),
  \[
  f(u, v) \leq c(u, v).
  \]

- **Flow conservation:** For all \( u \in V - \{s, t\} \),
  \[
  \sum_{v \in V} f(u, v) = 0.
  \]

- **Skew symmetry:** For all \( u, v \in V \),
  \[
  f(u, v) = -f(v, u).
  \]

Proof (continued)

Obtaining the positive flow from the net flow

(\( \Leftarrow \)) Define

\[
  p(u, v) = \begin{cases} 
  f(u, v) & \text{if } f(u, v) > 0, \\
  0 & \text{if } f(u, v) \leq 0.
  \end{cases}
\]

- **Capacity constraint:** By definition, \( p(u, v) \geq 0 \).
  Since \( f(u, v) \leq c(u, v) \), it follows that \( p(u, v) \leq c(u, v) \).

- **Flow conservation:** If \( f(u, v) > 0 \), then \( f(v, u) < 0 \) so \( p(v, u) = 0 \).
  If \( f(u, v) \leq 0 \), then
  \[
  p(u, v) - p(v, u) = f(u, v) = f(u, v)
  \]
  by skew symmetry. Therefore,

  \[
  \sum_{v \in V} p(u, v) - \sum_{v \in V} p(v, u) = \sum_{v \in V} f(u, v)
  \]
Residual network

Definition. Let \( f \) be a flow on \( G = (V, E) \). The residual network \( G_f(V, E) \) is the graph with strictly positive residual capacities \( c_f(u, v) = c(u, v) - f(u, v) > 0 \).

Examples:

\[
G:\begin{array}{ccc}
A & B & C \\
\text{4} & \text{4} & \text{5} \\
\end{array}
\]
\[
G_f:\begin{array}{ccc}
A & B & C \\
\text{4} & \text{4} & \text{5} \\
\end{array}
\]

Lemma. \( |E_f| \leq 2|E| \).

Augmenting paths

Definition. Any path from \( s \) to \( t \) in \( G_f \) is an augmenting path in \( G \) with respect to \( f \).

The flow value can be increased along an augmenting path \( p \) by adding \( c_f(p) := \min \{ c_f(u, v) \mid (u, v) \in E \} \) to the net flow of each edge along \( p \).

This is called path augmentation.

Examples:

\[
G:\begin{array}{ccc}
A & B & C \\
\text{3} & \text{5} & \text{7} \\
\end{array}
\]

Ford-Fulkerson max-flow algorithm

Start: \( f[u, v] = 0 \) for all \( u, v \in V \)

While (1) {
  construct \( G_f \)
  if an augmenting path \( p \) in \( G_f \) exists then
    augment \( f \) by \( c_f(p) \) //Any path would do
    else exit
}

Example 3 – maximum matching

Note – flow conservation is preserved.
Another example - Matching

|f| = 1

A B

G: 1:1
0:1 0:1 0:1 0:1
1:1
1:1
G: 1:1
0:1 0:1 0:1 0:1
1:1
|
|f| = 2

G:

Notation

Definition. The value of a flow \( f \), denoted by \(|f|\), is given by

\[ |f| = \sum_{v \in V} f(s,v) = f(s,V). \]

Implicit summation notation: A set used in an arithmetic formula represents a sum over the elements of the set.

Example — flow conservation:

\[ f(u,V) = \sum_{v \in V} f(u,v) = 0 \]

for all \( u \in V - \{s,t\} \).

More definitions

\[ f(X,Y) = \sum_{u \in X} \sum_{v \in Y} f(u,v) \]

More properties of flow

Lemma:
1. If \( X \) does not contain \( s \) nor \( t \), then \( f(X,V) = 0 \)

   Proof: \( f(X,V) = \sum_{u \in X} \sum_{v \in V} f(u,v) = \sum_{u \in X} 0 \).

2. If \( A,B \) are disjoint sets of vertices, and \( X \) is another set, then

   \[ f(A \cup B, X) = f(A,X) + f(B,X) \]

Note (property *): \( f(A,X) = f(A \cup B, X) - f(B,X) \)

And more properties of flow...

Lemma (Property #):
For every set \( X \) of vertices

\[ f(X,X) = 0 \]

Proof: \( f(X,X) = \sum_{u \in X} \sum_{v \in X} f(u,v) \), and if \( f(u,v) \) appears in the summation, then \( f(v,u) \) also appears in the summation, and \( f(v,u) = -f(u,v) \).

Simple properties of flow

Recall: \(|f| = f(s,V) = \sum_{v \in V} f(s,v)\)

Theorem. \(|f| = f(V,t)\).

Proof.

\[ |f| = f(s,V) = f(V,V-s) \text{ (Property *)} \]
\[ = f(V,V-s) \text{ (Property #)} \]
\[ = f(V,t) + f(V,V-s-t) \text{ (Case 2)} \]
\[ = f(V,t) \text{ (Case 1)} \]
Introduction to Algorithms, Lecture 22
December 5, 2001

Flow into the sink

Cuts

Definitions. A cut \((S, T)\) of a flow network \(G = (V, E)\) is a partition of \(V\) such that \(s \in S\) and \(t \in T\).

If \(f\) is a flow on \(G\), then the flow across the cut is \(f(S, T)\).

Another characterization of flow value

Lemma. For any flow \(f\) and any cut \((S, T)\), we have \(|f| = f(S, V)\).

Proof: \(f(S, T) = f(s, V) + f(S-S, V) = f(s, V)\) (property *)

Upper bound on the maximum flow value

Theorem. The value of any flow no larger than the capacity of any cut: \(|f| \leq c(S, T)\).

Proof. \(|f| = f(S, T) = \sum_{u \in S} \sum_{v \in T} f(u, v) \leq \sum_{u \in S} \sum_{v \in T} c(u, v) = c(S, T)\)

Max-flow, min-cut theorem

Theorem. The following are equivalent:
1. \(|f| = c(S, T)\) for some cut \((S, T)\).
2. \(f\) is a maximum flow.
3. \(f\) admits no augmenting paths.

Proof. (1) \(\Rightarrow\) (2): Since \(|f| \leq c(S, T)\) for any cut \((S, T)\) (by the theorem from a few slides back), the assumption that \(|f| = c(S, T)\) implies that \(f\) is a maximum flow.

(2) \(\Rightarrow\) (3): If there were an augmenting path, the flow value could be increased, contradicting the maximality of \(f\).
(I) ⇒ (II): Define \( S = \{ v \in V \mid \) there exists a path in \( G_f \) from \( s \) to \( v \} \).
Let \( T = V - S \). Since \( f \) admits no augmenting paths, there is no path from \( s \) to \( t \) in \( G_f \).
Hence, \( s \in S \) and \( t \notin S \), so \( s \in T \).
Thus \( (S, T) \) is a cut. Consider any vertices \( u \in S \) and \( v \in T \).

Consider \( u \in S, v \in T \). We must have \( c_f(u, v) = 0 \), since if \( c_f(u, v) > 0 \), then \( v \in S \), not \( v \in T \) as assumed.
Thus, \( f(u, v) = c(u, v) \), since \( c_f(u, v) = c(u, v) - f(u, v) \).
Summing over all \( u \in S \) and \( v \in T \) yields \( f(S, T) = c(S, T) \), and since \( |f| = f(S, T) \), the theorem follows.
**Ford-Fulkerson max-flow algorithm**

**Algorithm:**
\[
\begin{align*}
  f[u, v] &\leftarrow 0 \text{ for all } u, v \in V \\
  \text{while an augmenting path } p \text{ in } G \text{ wrt } f \text{ exists} & \text{ do} \\
  \text{augment } f \text{ by } c_f(p) \\
\end{align*}
\]

*Can be slow:*

\[
G: \quad s \quad t
\]

- 1:10^9
- 0:1
- 1:10^9
- 1:10^9
- 1:10^9

**Edmonds-Karp algorithm**

Edmonds and Karp noticed that many people’s implementations of Ford-Fulkerson augment along a breadth-first augmenting path: a path with smallest number of edges in \(G_f\) from \(s\) to \(t\).

These implementations would always run relatively fast.

Since a breadth-first augmenting path can be found in \(O(|E|)\) time, their analysis, focuses on bounding the number of flow augmentations.

(In independent work, Dinic also gave polynomial-time bounds.)

**Runtime analysis of F&F-algorithm applied for matching**

- We saw that in each iteration of F&F algorithm, \(|f|\) increases by at least 1.
- Let \(|f^*|\) be the maximum value.
- How large can \(|f^*|\) be?

- Claim: \(|f^*| \leq \min|A|, |B|\) (why?)
- Runtime is \(O(|E|\min|A|, |B|) = O(|E||V|)\)
- Can be done in \(O(|E|^{1/2} \cdot |V|)\) (Dinic Algorithm)

**Ford-Fulkerson and matching**

Recall – we expressed the maximum matching problem as a network flow, but we can express the max flow as a matching, only if the flow is an *integer* flow.

However, this is always the case once using F&F algorithm: The flow along each edge is either 0 or 1.
Running time of Edmonds-Karp

- One can show that the number of flow augmentations (i.e., the number of iterations of the while loop) is $O(|V||E|)$.
- Breadth-first search runs in $O(|E|)$ time
- All other bookkeeping is $O(|V|)$ per augmentation.
⇒ The Edmonds-Karp maximum-flow algorithm runs in $O(|V||E|^2)$ time.

Best to date

- The asymptotically fastest algorithm to date for maximum flow, due to King, Rao, and Tarjan, runs in $O(V E \log_{\log(V)} V)$ time.
- If we allow running times as a function of edge weights, the fastest algorithm for maximum flow, due to Goldberg and Rao, runs in time $O(\min\{V^{2/3}, E^{1/2}\} \cdot E \cdot V \cdot (V^2/2 + 2) \cdot \log C)$, where $C$ is the maximum capacity of any edge in the graph.