Flow Networks and the Min-Cut-Max-Flow Theorem

Peter Lammich and S. Reza Sefidgar

August 16, 2018

Abstract

We present a formalization of flow networks and the Min-Cut-Max-Flow theorem. Our formal proof closely follows a standard textbook proof, and is accessible even without being an expert in Isabelle/HOL—the interactive theorem prover used for the formalization.
Contents

1 Introduction 3

2 Flows, Cuts, and Networks 3
   2.1 Definitions 3
      2.1.1 Flows 3
      2.1.2 Cuts 4
      2.1.3 Networks 4
      2.1.4 Networks with Flows and Cuts 6
   2.2 Properties 7
      2.2.1 Flows 7
      2.2.2 Networks 8
      2.2.3 Networks with Flow 9

3 Residual Graph 10
   3.1 Definition 10
   3.2 Properties 11

4 Augmenting Flows 14
   4.1 Augmentation of a Flow 14
   4.2 Augmentation yields Valid Flow 15
      4.2.1 Capacity Constraint 15
   4.3 Value of the Augmented Flow 16

5 Augmenting Paths 16
   5.1 Definitions 17
   5.2 Augmenting Flow is Valid Flow 17
   5.3 Value of Augmenting Flow is Residual Capacity 18

6 The Ford-Fulkerson Theorem 18
   6.1 Net Flow 18
   6.2 Ford-Fulkerson Theorem 19
   6.3 Corollaries 19
1 Introduction

Computing the maximum flow of a network is an important problem in graph theory. Many other problems, like maximum-bipartite-matching, edge-disjoint-paths, circulation-demand, as well as various scheduling and resource allocating problems can be reduced to it. The Ford-Fulkerson method [3] describes a class of algorithms to solve the maximum flow problem. It is based on a corollary of the Min-Cut-Max-Flow theorem [3, 2], which states that a flow is maximal iff there exists no augmenting path.

In this chapter, we present a formalization of flow networks and prove the Min-Cut-Max-Flow theorem, closely following the textbook presentation of Cormen et al. [1]. We have used the Isar [4] proof language to develop human-readable proofs that are accessible even to non-Isabelle experts.

2 Flows, Cuts, and Networks

theory Network
imports Graph
begin

In this theory, we define the basic concepts of flows, cuts, and (flow) networks.

2.1 Definitions

2.1.1 Flows

An \( s-t \) preflow on a graph is a labeling of the edges with values from a linearly ordered integral domain, such that:

- **capacity constraint** the flow on each edge is non-negative and does not exceed the edge’s capacity;
- **non-deficiency constraint** for all nodes except \( s \) and \( t \), the incoming flow greater or equal to the outgoing flow.

**type-synonym** 'capacity flow = edge ⇒ 'capacity

**locale** Preflow = Graph c for c :: 'capacity::linordered-idom graph +

fixes s t :: node

fixes f :: 'capacity flow

assumes capacity-const: \( \forall e. \ 0 \leq f e \land f e \leq c e \)

assumes no-deficient-nodes: \( \forall v \in V - \{s,t\}. \)

(\( \sum e \in \text{outgoing } v. f e \)) \leq (\( \sum e \in \text{incoming } v. f e \))

begin
An s-t flow on a graph is a preflow that has no active nodes except source and sink, where a node is active iff it has more incoming flow than outgoing flow.

locale Flow = Preflow c s t f
for c :: 'capacity::linordered-idom graph
and s t :: node
and f +
assumes no-active-nodes:
∀ v ∈ V − {s, t}. (∑ e ∈ incoming v. f e) ≥ (∑ e ∈ outgoing v. f e)
begin
For a flow, inflow equals outflow for all nodes except sink and source. This is called conservation.

lemma conservation-const:
∀ v ∈ V − {s, t}. (∑ e ∈ incoming v. f e) = (∑ e ∈ outgoing v. f e)
⟨proof⟩
The value of a flow is the flow that leaves s and does not return.

definition val :: 'capacity
where val ≡ (∑ e ∈ outgoing s. f e) − (∑ e ∈ incoming s. f e)
end

locale Finite-Preflow = Preflow c s t f + Finite-Graph c
for c :: 'capacity::linordered-idom graph and s t f

locale Finite-Flow = Flow c s t f + Finite-Preflow c s t f
for c :: 'capacity::linordered-idom graph and s t f

2.1.2 Cuts

A cut is a partitioning of the nodes into two sets. We define it by just specifying one of the partitions. The other partition is implicitly given by the remaining nodes.

type-synonym cut = node set

locale Cut = Graph +
fixes k :: cut
assumes cut-ss-V: k ⊆ V

2.1.3 Networks

A network is a finite graph with two distinct nodes, source and sink, such that all edges are labeled with positive capacities. Moreover, we assume that

- The source has no incoming edges, and the sink has no outgoing edges.
• There are no parallel edges, i.e., for any edge, the reverse edge must not be in the network.

• Every node must lay on a path from the source to the sink.

Notes on the formalization
• We encode the graph by a mapping $c$, such that $c(u, v)$ is the capacity of edge $(u, v)$, or 0, if there is no edge from $u$ to $v$. Thus, in the formalization below, we only demand that $c(u, v) \geq 0$ for all $u$ and $v$.

• We only demand the set of nodes reachable from the source to be finite. Together with the constraint that all nodes lay on a path from the source, this implies that the graph is finite.

locale Network = Graph $c$ for $c$ :: 'capacity::linordered-idom graph +
fixes $s$ $t$ :: node
assumes s-node[simp, intro!]: $s \in V$
assumes t-node[simp, intro!]: $t \in V$
assumes s-not-t[simp, intro!]: $s \neq t$
assumes cap-non-negative: $\forall u \ v. \ c(u, v) \geq 0$
assumes no-incoming-s: $\forall u. \ (u, s) \notin E$
assumes no-outgoing-t: $\forall u. \ (t, u) \notin E$
assumes no-parallel-edge: $\forall u \ v. \ (u, v) \in E \implies (v, u) \notin E$
assumes nodes-on-st-path: $\forall v \in V. \ connected\ s\ v \land connected\ v\ t$
assumes finite-reachable: finite (reachableNodes $s$)
begin

Edges have positive capacity

lemma edge-cap-positive: $(u,v)\in E \implies c(u,v) > 0$
-proof

The network constraints implies that all nodes are reachable from the source node

lemma reachable-is-V[simp]: reachableNodes $s = V$
-proof

Thus, the network is actually a finite graph.

sublocale Finite-Graph
-proof

Our assumptions imply that there are no self loops

lemma no-self-loop: $\forall u. \ (u, u) \notin E$
-proof

lemma adjacent-not-self[simp, intro!]: $v \notin adjacent-nodes$ $v$
A flow is maximal, if it has a maximal value

\[ \text{isMaxFlow} \quad \text{::} \quad \text{flow} \rightarrow \text{bool} \]

where
\[ \text{isMaxFlow} \ f \equiv \text{Flow} \ c \ s \ t \ f \land \ (\forall f'. \text{Flow} \ c \ s \ t \ f' \rightarrow \text{Flow} \ . \text{val} \ c \ s \ f' \leq \text{Flow} \ . \text{val} \ c \ s \ f) \]

\[ \text{is-max-flow-val} \quad \text{::} \quad \text{flow} \rightarrow \exists f. \text{isMaxFlow} \ f \land f \ = \text{Flow} \ . \text{val} \ c \ s \ f \]

}\[\text{(proof)}\]

**Lemma:** \( t \not= s \) \[\text{(proof)}\]

The excess of a node is the difference between incoming and outgoing flow.

\[ \text{excess} \quad \text{::} \quad \text{capacity} \ . \text{flow} \rightarrow \text{node} \rightarrow \text{capacity} \]

where
\[ \text{excess} \ f \ v \equiv (\sum e \in \text{incoming} \ v \ . \ f \ e) - (\sum e \in \text{outgoing} \ v \ . \ f \ e) \]

\[\text{end}\]

### 2.1.4 Networks with Flows and Cuts

For convenience, we define locales for a network with a fixed flow, and a network with a fixed cut

**locale** \( \text{NPreflow} \)

\[ \text{Network} \ c \ s \ t + \text{Preflow} \ c \ s \ t \ f \]

\[\text{for} \ c :: \text{capacity} \\quad \text{and} \quad \text{linordered-idom} \ \text{graph} \quad \text{and} \quad \text{s} \ \text{t} \ \text{f} \]

**begin**

\[\text{end}\]

**locale** \( \text{NFlow} \)

\[ \text{NPreflow} \ c \ s \ t \ f + \text{Flow} \ c \ s \ t \ f \]

\[\text{for} \ c :: \text{capacity} \\quad \text{and} \quad \text{linordered-idom} \ \text{graph} \quad \text{and} \quad \text{s} \ \text{t} \ \text{f} \]

**lemma** \( \text{in Network} \) \( \text{isMaxFlow-alt} \):

\[ \text{isMaxFlow} \ f \longleftrightarrow \text{NFlow} \ c \ s \ t \ f \land \ (\forall f'. \text{NFlow} \ c \ s \ t \ f' \rightarrow \text{Flow} \ . \text{val} \ c \ s \ f' \leq \text{Flow} \ . \text{val} \ c \ s \ f) \]

\[\text{(proof)}\]

A cut in a network separates the source from the sink

**locale** \( \text{NCut} \)

\[ \text{Network} \ c \ s \ t + \text{Cut} \ c \ k \]

\[\text{for} \ c :: \text{capacity} \\quad \text{and} \quad \text{linordered-idom} \ \text{graph} \quad \text{and} \quad \text{s} \ \text{t} \ \text{k} + \]

\[\text{assumes} \ s-\text{in-cut} :: \ s \ \in \ k \]

\[\text{assumes} \ t-\text{ni-cut} :: t \ \notin \ k \]

**begin**

\[\text{end}\]

The capacity of the cut is the capacity of all edges going from the source’s side to the sink’s side.

**definition** \( \text{cap} :: \text{capacity} \)
where \(\text{cap} \equiv (\sum e \in \text{outgoing}' \ k \ e)\)

end

A minimum cut is a cut with minimum capacity.

**definition** isMinCut :: - graph \(\Rightarrow\) nat \(\Rightarrow\) nat \(\Rightarrow\) cut \(\Rightarrow\) bool

**where** isMinCut c s t k \(\equiv\) NCut c s t k \(\land\)

\((\forall k'. \ NCut c s t k' \rightarrow NCut.cap c k \leq NCut.cap c k')\)

### 2.2 Properties

#### 2.2.1 Flows

**context** Preflow

**begin**

Only edges are labeled with non-zero flows

**lemma** zero-flow-simp|simp|:

\((u,v) \notin E \Rightarrow f(u,v) = 0\)

(proof)

**lemma** f-non-negative:

\(0 \leq f\ e\)

(proof)

**lemma** sum-f-non-negative:

\(\sum f\ X \geq 0\)

(proof)

**end** — Preflow

**context** Flow

**begin**

We provide a useful equivalent formulation of the conservation constraint.

**lemma** conservation-const-pointwise:

assumes \(\forall v \in V - \{s,t\}\)

shows \((\sum v \in E'\{u\}, f(u,v)) = (\sum v \in E^{-1}\{u\}, f(v,u))\)

(proof)

The value of the flow is bounded by the capacity of the outgoing edges of the source node

**lemma** val-bounded:

\(-\)(\(\sum e \in \text{incoming}\ s \ e\)) \(\leq\) val

val \(\leq\) \(\sum e \in \text{outgoing}\ s \ e\)

(proof)

**end** — Flow

Introduce a flow via the conservation constraint

**lemma** (in Graph) intro-Flow:
assumes \( \forall e. 0 \leq f e \land f e \leq c e \)
assumes \( \forall v \in V = \{s, t\}. (\sum e \in \text{incoming } v. f e) = (\sum e \in \text{outgoing } v. f e) \)
shows \( \text{Flow } c s t f \)

\begin{verbatim}
context Finite-Preflow
begin

The summation of flows over incoming/outgoing edges can be extended to a
summation over all possible predecessor/successor nodes, as the additional
flows are all zero.

\textbf{lemma} \( \text{sum-outgoing-alt-flow}: \)
fixes \( g :: \text{edge } \Rightarrow 'c \text{capacity} \)
assumes \( u \in V \)
shows \( (\sum e \in \text{outgoing } u. f e) = (\sum v \in V. f (u,v)) \)
\( \langle \text{proof} \rangle \)

\textbf{lemma} \( \text{sum-incoming-alt-flow}: \)
fixes \( g :: \text{edge } \Rightarrow 'c \text{capacity} \)
assumes \( u \in V \)
shows \( (\sum e \in \text{incoming } u. f e) = (\sum v \in V. f (v,u)) \)
\( \langle \text{proof} \rangle \)
end — Finite Preflow
\end{verbatim}

\subsection{2.2.2 Networks}

\textbf{context} \( \text{Network} \)
\textbf{begin}

\textbf{lemmas} \( \text{[simp]} = \text{no-incoming-s no-outgoing-t} \)

\textbf{lemma} \( \text{incoming-s-empty[simp]: incoming } s = {} \)
\( \langle \text{proof} \rangle \)

\textbf{lemma} \( \text{outgoing-t-empty[simp]: outgoing } t = {} \)
\( \langle \text{proof} \rangle \)

\textbf{lemma} \( \text{cap-positive: } e \in E \implies c e > 0 \)
\( \langle \text{proof} \rangle \)

\textbf{lemma} \( \text{V-not-empty: } V \neq \{\} \langle \text{proof} \rangle \)
\textbf{lemma} \( \text{E-not-empty: } E \neq \{\} \langle \text{proof} \rangle \)

\textbf{lemma} \( \text{card-V-ge2: card } V \geq 2 \)
\( \langle \text{proof} \rangle \)

\textbf{lemma} \( \text{zero-is-flow: Flow } c s t (\lambda-. 0) \)
proof

lemma max-flow-val-unique:
    [is-max-flow-val fv1; is-max-flow-val fv2] ⇒ fv1 = fv2

end — Network

2.2.3 Networks with Flow

context NPreflow begin

sublocale Finite-Preflow ⟨proof⟩

As there are no edges entering the source/leaving the sink, also the corresponding flow values are zero:

lemma no-inflow-s: ∀ e ∈ incoming s. f e = 0 (is ?thesis)
⟨proof⟩

lemma no-outflow-t: ∀ e ∈ outgoing t. f e = 0
⟨proof⟩

For an edge, there is no reverse edge, and thus, no flow in the reverse direction:

lemma zero-rev-flow-simp[simp]: (u, v) ∈ E ⇒ f(v, u) = 0
⟨proof⟩

lemma excess-non-negative: ∀ v ∈ V − {s, t}. excess f v ≥ 0
⟨proof⟩

lemma excess-nodes-only: excess f v > 0 ⇒ v ∈ V
⟨proof⟩

lemma excess-non-negative': ∀ v ∈ V − {s}. excess f v ≥ 0
⟨proof⟩

lemma excess-s-non-pos: excess f s ≤ 0
⟨proof⟩

end — Network with preflow

context NFlow begin

sublocale Finite-Preflow ⟨proof⟩

There is no outflow from the sink in a network. Thus, we can simplify the definition of the value:

corollary val-alt: val = (∑ e ∈ outgoing s. f e)
3 Residual Graph

theory Residual-Graph
imports Network
begin

In this theory, we define the residual graph.

3.1 Definition

The residual graph of a network and a flow indicates how much flow can be effectively pushed along or reverse to a network edge, by increasing or decreasing the flow on that edge:

\[ \text{residualGraph} : \text{graph} \Rightarrow \text{flow} \Rightarrow \text{graph} \]

where

\[ \text{residualGraph} c f \equiv \lambda(u, v). \]

- if \((u, v) \in \text{Graph.E} c\) then
  \[ c(u, v) - f(u, v) \]
- else if \((v, u) \in \text{Graph.E} c\) then
  \[ f(v, u) \]
- else
  \[ 0 \]

context Network begin

abbreviation cf-of \equiv \text{residualGraph} c
abbreviation cfE-of f \equiv \text{Graph.E} (cf-of f)

The edges of the residual graph are either parallel or reverse to the edges of the network.

lemma cfE-of-ss-invE: cfE-of cf \subseteq E \cup E^{-1}
⟨proof⟩

lemma cfE-of-ss-VxV: cfE-of f \subseteq V \times V
⟨proof⟩

lemma cfE-of-finite[simp, intro!]: finite (cfE-of f)
⟨proof⟩

lemma cf-no-self-loop: (u, u) \notin cfE-of f
⟨proof⟩
Let’s fix a network with a preflow $f$ on it

**context** $N$Preflow

**begin**

We abbreviate the residual graph by $cf$.

**abbreviation** $cf \equiv \text{residualGraph } c f$

**sublocale** $cf$: Graph $cf$ ($proof$)

**lemmas** $cf$-def $\equiv$ residualGraph-def[$of c f$]

### 3.2 Properties

**lemmas** $cfE$-ss-invE $\equiv$ $cfE-of$-ss-invE[$of f$]

The nodes of the residual graph are exactly the nodes of the network.

**lemma** res$V$-net$V$[$simp$]: $cf$. $V$ $\equiv$ $V$

($proof$)

Note, that Isabelle is powerful enough to prove the above case distinctions completely automatically, although it takes some time:

**lemma** $cf$. $V$ $\equiv$ $V$

($proof$)

As the residual graph has the same nodes as the network, it is also finite:

**sublocale** $cf$: Finite-Graph $cf$

($proof$)

The capacities on the edges of the residual graph are non-negative

**lemma** res$E$-nonNegative: $cf$ $e$ $\geq$ $0$

($proof$)

Again, there is an automatic proof

**lemma** $cf$ $e$ $\geq$ $0$

($proof$)

All edges of the residual graph are labeled with positive capacities:

**corollary** res$E$-positive: $e$ $\in$ $cf$. $E$ $\Rightarrow$ $cf$ $e$ $>$ $0$

($proof$)

**lemma** reverse-flow: Preflow $cf$ $s$ $t$ $f'$ $\Rightarrow$ $\forall (u, v) \in E$. $f'$ $(v, u) \leq f$ $(u, v)$

($proof$)

**definition** (in Network) flow-of-cf $cf$ $e$ $\equiv$ ($if$ ($e$ $\in$ $E$) then $c$ $e$ $-$ $cf$ $e$ else $0$)
lemma (in NPreflow) E-ss-cfinvE: $E \subseteq \text{Graph}.E \cup (\text{Graph}.E)_{-1}$

Nodes with positive excess must have an outgoing edge in the residual graph. Intuitively: The excess flow must come from somewhere.

lemma active-has-cf-outgoing: $\text{excess } f.u > 0 \implies \text{cf.outgoing } u \neq \{\}$

end — Network with preflow

locale RPreGraph — Locale that characterizes a residual graph of a network

= Network +
fixes cf
assumes EX-RPG: $\exists f. \text{NPreflow } c.s.t.f \wedge \text{cf} = \text{residualGraph } c.f$
begin

lemma this-loc-rpg: RPreGraph c s t cf

definition $f \equiv \text{flow-of-cf } cf$

lemma f-unique:
assumes NPreflow c s t f' 
assumes A: $\text{cf} = \text{residualGraph } c.f'$
shows $f' = f$

lemma is-NPreflow: NPreflow c s t (flow-of-cf cf)

sublocale f: NPreflow c s t f (proof)

lemma rg-is-cf[simp]: residualGraph c f = cf

lemma rg-fo-inv[simp]: residualGraph c (flow-of-cf cf) = cf

sublocale cf: Graph cf (proof)

lemma resV-netV[simp]: $\text{cf}.V = V$

sublocale cf: Finite-Graph cf
\begin{proof}

\textbf{lemma} \ E-ss-cfinvE: \ E \subseteq \ cf \cdot E \cup \ cf \cdot E^{-1}

\textbf{proof}

\textbf{lemma} \ cfE-ss-invE: \ cf \cdot E \subseteq E \cup E^{-1}

\textbf{proof}

\textbf{lemma} \ resE-nonNegative: \ cf \cdot e \geq 0

\textbf{proof}

\end

\begin{context} \textit{NPreflow} \begin{proof}

\textbf{lemma} \ is-RPreGraph: \ RPreGraph \ c \ s \ t \ cf

\textbf{proof}

\textbf{lemma} \ fo-rg-inv: \ flow-of-cf \ cf = f

\textbf{proof}

\end

\begin{proof}

\textbf{lemma} \ (in \ \textit{NPreflow}) \ flow-of-cf \: (\text{residualGraph} \ c \ f) = f

\textbf{proof}

\end

\begin{locale} \textit{RGraph} \ — \ Locale \ that \ characterizes \ a \ residual \ graph \ of \ a \ network \ = \ Network +

\textbf{fixes} \ cf

\textbf{assumes} \ EX-RG: \ \exists f. \ NFlow \ c \ s \ t \ f \wedge \ cf = \text{residualGraph} \ c \ f

\begin{proof}

\textbf{sublocale} \ RPreGraph

\textbf{proof}

\textbf{lemma} \ this-loc: \ RGraph \ c \ s \ t \ cf

\textbf{proof}

\textbf{lemma} \ this-loc-rpg: \ RPreGraph \ c \ s \ t \ cf

\textbf{proof}

\textbf{lemma} \ is-NFlow: \ NFlow \ c \ s \ t \ (\text{flow-of-cf} \ cf)

\textbf{proof}

\textbf{sublocale} \ f: \ NFlow \ c \ s \ t \ f \ (proof)

\end

\begin{context} \textit{NFlow} \begin{proof}

\end

\end

\end

\end
The value of the flow can be computed from the residual graph.

Theorem 1 (in RPreGraph) maxflow-imp-rgraph:
assumes isMaxFlow (flow-of cf cf)
shows RGraph c s t cf
(proof)

end — Theory

4 Augmenting Flows

Theorem Augmenting-Flow
imports Residual-Graph
begin

In this theory, we define the concept of an augmenting flow, augmentation with a flow, and show that augmentation of a flow with an augmenting flow yields a valid flow again.

We assume that there is a network with a flow \( f \) on it

context NFlow
begin

4.1 Augmentation of a Flow

The flow can be augmented by another flow, by adding the flows of edges parallel to edges in the network, and subtracting the edges reverse to edges in the network.

Definition augment :: 'capacity flow ⇒ 'capacity flow
where augment \( f' \) ≡ \( \lambda (u, v). \)
if \( (u, v) \in E \) then
\( f (u, v) + f' (u, v) - f' (v, u) \)
else
0

We define a syntax similar to Cormen et al.:

Abbreviation (input) augment-syntax (infix ↑ 55)
where \( \forall f', f↑f' \equiv NFlow.augment c f f' \)
such that we can write \( f↑f' \) for the flow \( f \) augmented by \( f' \).
4.2 Augmentation yields Valid Flow

We show that, if we augment the flow with a valid flow of the residual graph, the augmented flow is a valid flow again, i.e. it satisfies the capacity and conservation constraints:

context
— Let the residual flow \( f' \) be a flow in the residual graph

fixes \( f' :: 'capacity flow \)

assumes \( f'-flow: Flow \ of \ s \ t \ f' \)

begin

interpretation \( f' \): Flow \ of \ s \ t \ f' \)

4.2.1 Capacity Constraint

First, we have to show that the new flow satisfies the capacity constraint:

\begin{align*}
\text{lemma} \ & \text{augment-flow-presv-cap:} \\
& \text{shows} \ 0 \leq (f \uplus f')(u,v) \wedge (f \uplus f')(u,v) \leq c(u,v)
\end{align*}

(proof) 
\begin{align*}
\text{lemma} \ & \text{split-rflow-incoming:} \\
& \left( \sum_{v \in cf.E^{-1}} f'(v,u) \right) - \left( \sum_{v \in cf.E^{-1}} f'(v,u) \right) = \left( \sum_{v \in cf.E^{-1}} f'(v,u) \right) - \left( \sum_{v \in cf.E^{-1}} f'(v,u) \right)
\end{align*}
(is \( LHS = ?RHS \))

(proof)

For proving the conservation constraint, let’s fix a node \( u \), which is neither the source nor the sink:

context
— fixes \( u :: node \)

assumes \( U-ASM: u \in V \setminus \{s,t\} \)

begin

We first show an auxiliary lemma to compare the effective residual flow on incoming network edges to the effective residual flow on outgoing network edges.

Intuitively, this lemma shows that the effective residual flow added to the network edges satisfies the conservation constraint.

private lemma \text{flow-summation-aux:}

shows \( \left( \sum_{v \in cf.E^{-1}} f'(v,u) \right) - \left( \sum_{v \in cf.E^{-1}} f'(v,u) \right) = \left( \sum_{v \in cf.E^{-1}} f'(v,u) \right) - \left( \sum_{v \in cf.E^{-1}} f'(v,u) \right) \)
(is \( LHS = ?RHS \) is \( ?A - ?B = ?RHS \))

(proof)

Finally, we are ready to prove that the augmented flow satisfies the conservation constraint:

\begin{align*}
\text{lemma} \ & \text{augment-flow-presv-con:} \\
& \text{shows} \ (\sum_{e \in outgoing \ u \ \text{augment } f' \ e}) = (\sum_{e \in incoming \ u \ \text{augment } f' \ e})
\end{align*}
(is \( LHS = ?RHS \))

15
Note that we tried to follow the proof presented by Cormen et al. [1] as closely as possible. Unfortunately, this proof generalizes the summation to all nodes immediately, rendering the first equation invalid. Trying to fix this error, we encountered that the step that uses the conservation constraints on the augmenting flow is more subtle as indicated in the original proof. Thus, we moved this argument to an auxiliary lemma.

As main result, we get that the augmented flow is again a valid flow.

**corollary augment-flow-presv:** \( \text{Flow} \ c \ s \ t \ (f \uparrow f') \)

**4.3 Value of the Augmented Flow**

Next, we show that the value of the augmented flow is the sum of the values of the original flow and the augmenting flow.

**lemma augment-flow-value:** \( \text{Flow}.\val c \ s \ (f \uparrow f') = \val + \text{Flow}.\val c f f' \)

Note, there is also an automatic proof. When creating the above explicit proof, this automatic one has been used to extract meaningful subgoals, abusing Isabelle as a term rewriter.

**5 Augmenting Paths**

**theory Augmenting-Path**

**imports** Residual-Graph

**begin**

We define the concept of an augmenting path in the residual graph, and the residual flow induced by an augmenting path.

We fix a network with a preflow \( f \) on it.

**context NPreflow**

**begin**
5.1 Definitions

An augmenting path is a simple path from the source to the sink in the residual graph:

**definition** isAugmentingPath :: path ⇒ bool

**where** isAugmentingPath p ≡ cf.isSimplePath s p t

The residual capacity of an augmenting path is the smallest capacity annotated to its edges:

**definition** resCap :: path ⇒ 'capacity

**where** resCap p ≡ Min {cf e | e. e ∈ set p}

**lemma** resCap-alt: resCap p = Min (cf set p)

— Useful characterization for finiteness arguments

(proof)

An augmenting path induces an augmenting flow, which pushes as much flow as possible along the path:

**definition** augmentingFlow :: path ⇒ 'capacity flow

**where** augmentingFlow p ≡ λ(u, v).

if (u, v) ∈ (set p) then
    resCap p
else
    0

5.2 Augmenting Flow is Valid Flow

In this section, we show that the augmenting flow induced by an augmenting path is a valid flow in the residual graph.

We start with some auxiliary lemmas.

The residual capacity of an augmenting path is always positive.

**lemma** resCap-gzero-aux: cf.isPath s p t ⇒ 0<resCap p

(proof)

**lemma** resCap-gzero: isAugmentingPath p ⇒ 0<resCap p

(proof)

As all edges of the augmenting flow have the same value, we can factor this out from a summation:

**lemma** sum-augmenting-alt:

**assumes** finite A

**shows** (∑ e ∈ A. (augmentingFlow p) e) = resCap p * of-nat (card (A∩set p))

(proof)
lemma augFlow-resFlow: isAugmentingPath p \implies Flow cf s t (augmentingFlow p)
⟨proof⟩

5.3 Value of Augmenting Flow is Residual Capacity

Finally, we show that the value of the augmenting flow is the residual capacity of the augmenting path

lemma augFlow-val:
  isAugmentingPath p \implies Flow.val cf s (augmentingFlow p) = resCap p
⟨proof⟩

end — Network with flow
end — Theory

6 The Ford-Fulkerson Theorem

theory Ford-Fulkerson
imports Augmenting-Flow Augmenting-Path
begin

In this theory, we prove the Ford-Fulkerson theorem, and its well-known corollary, the min-cut max-flow theorem.

We fix a network with a flow and a cut

locale NFlowCut = NFlow c s t f + NCut c s t k
  for c :: 'capacity, linordered-idom graph and s t f k
begin

lemma finite-k[simp, intro!]: finite k
⟨proof⟩

6.1 Net Flow

We define the net flow to be the amount of flow effectively passed over the cut from the source to the sink:

definition netFlow :: 'capacity
  where netFlow \equiv (\sum e \in outgoing' k. f e) - (\sum e \in incoming' k. f e)

We can show that the net flow equals the value of the flow. Note: Cormen et al. [1] present a whole page full of summation calculations for this proof, and our formal proof also looks quite complicated.

lemma flow-value: netFlow = val
⟨proof⟩

The value of any flow is bounded by the capacity of any cut. This is intuitively clear, as all flow from the source to the sink has to go over the cut.
6.2 Ford-Fulkerson Theorem
context $NFlow$ begin

We prove three auxiliary lemmas first, and the state the theorem as a corollary.

lemma fofu-I-II: $\text{isMaxFlow } f \implies \neg (\exists \ p. \ \text{isAugmentingPath } p)$

lemma fofu-II-III:
$\neg (\exists \ p. \ \text{isAugmentingPath } p) \implies \exists k'. \ \text{NCut } c \ s \ t \ k' \land \text{val} = \text{NCut.cap } c \ k'$

lemma fofu-III-I:
$\exists k. \ \text{NCut } c \ s \ t \ k \land \text{val} = \text{NCut.cap } c \ k \implies \text{isMaxFlow } f$

Finally we can state the Ford-Fulkerson theorem:

theorem ford-fulkerson: shows
$\text{isMaxFlow } f \iff \neg \exists \text{ isAugmentingPath}$ and $\neg \exists \text{ isAugmentingPath} \iff \exists k. \ \text{NCut } c \ s \ t \ k \land \text{val} = \text{NCut.cap } c \ k$

6.3 Corollaries

In this subsection we present a few corollaries of the flow-cut relation and the Ford-Fulkerson theorem.

The outgoing flow of the source is the same as the incoming flow of the sink. Intuitively, this means that no flow is generated or lost in the network, except at the source and sink.

corollary inflow-t-outflow-s:
$(\sum e \in \text{incoming } t. \ f e) = (\sum e \in \text{outgoing } s. \ f e)$

As an immediate consequence of the Ford-Fulkerson theorem, we get that there is no augmenting path if and only if the flow is maximal.

corollary noAugPath-iff-maxFlow: $(\exists p. \ \text{isAugmentingPath } p) \iff \text{isMaxFlow } f$

end — Network with flow
The value of the maximum flow equals the capacity of the minimum cut

**corollary** (in Network) \( \text{maxFlow-minCut: } [\text{isMaxFlow}\; f; \text{isMinCut}\; c\; s\; t\; k] \Rightarrow \text{Flow}\.\text{val}\; c\; s\; f = \text{NCut}\.\text{cap}\; c\; k \) 

end — Theory

**References**


