VU Algorithmics

Part IV: Network Flows

Bin Hu, Günther Raidl, Mario Ruthmair

Algorithms and Data Structures Group
Institute of Computer Graphics and Algorithms
Vienna University of Technology

WS 2013/14

Topics of this part

- Maximum Flow Basics
- Ford-Fulkerson Maximum Flow Algorithm
- Preflow-Push Maximum Flow Algorithm
- Networks with Lower Capacity Bounds
- Minimum Cost Flow Problem

Maximum Flows in Networks

Maximum Flows: Introduction

Maximum flow problem arises in a wide variety of situations:

- Transport of petroleum products in a pipeline network.
- Transmission of data between two stations in a telecommunication network.
- . . .
- Subproblem in the solution of other, more difficult network problems, e.g. minimum cost flow.

Literature:
R.K. Ahuja, T.L. Magnanti, J.B. Orlin: *Network Flows*, Prentice Hall, 1993

Maximum Flows: Introduction

Definition 1 (Flow Network)

A flow network is a 5-tuple $\mathcal{N} = (V, A, \varsigma, s, t)$, with (V, A) being a directed graph with node set V and arc set A, a function $\varsigma : A \to \mathbb{R}_0^+$, and two nodes $s, t \in V, s \neq t$. Function ς associates nonnegative capacities to each arc (u, v), node s is called the source, node t the target or sink.

Extension: $\varsigma(a) = 0 \qquad \forall$ arcs $a \in (V \times V) \setminus A$.

Maximum Flows: Introduction

Assumption

The network $\mathcal{N}$ is connected
(guaranteed by the extension $\varsigma(a) = 0 \quad \forall a \in (V \times V) \setminus A$).

Assumption

The network $\mathcal{N}$ does not contain a directed path from s to t composed only of infinite capacity arcs.

Assumption

The network $\mathcal{N}$ does not contain parallel arcs (i.e., two or more arcs with the same tail and head nodes).

Maximum Flows: Introduction

Definition 2 (Flow)

A flow is a real function $f : V \times V \to \mathbb{R}$ with the following three properties:

1. Skew symmetry (asymmetry): $f(u, v) = -f(v, u) \qquad \forall u, v \in V$
2. Capacity constraints: $f(u, v) \leq \varsigma(u, v) \qquad \forall u, v \in V$
3. Flow conservation: $\sum_{v \in V} f(u, v) = f(u, V) = 0 \qquad \forall u \in V \setminus \{s, t\}$

Note: $f(A, B) = \sum_{u \in A} \sum_{v \in B} f(u, v)$; $f(\{u\}, V) = f(u, V)$

$f(u, v)$ is the *net flow* from node u to node v:
real flow of 4 units from u to v and of 3 units from v to u then the net flow $f(u, v) = 1$ and $f(v, u) = -1$.

Maximum Flows: Introduction

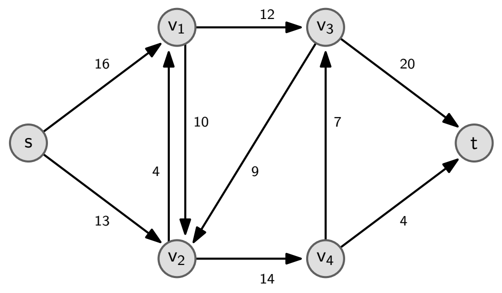


Figure: A flow network $\mathcal{N}$ with associated arc capacities $\varsigma(u, v)$.

Maximum Flows: Introduction

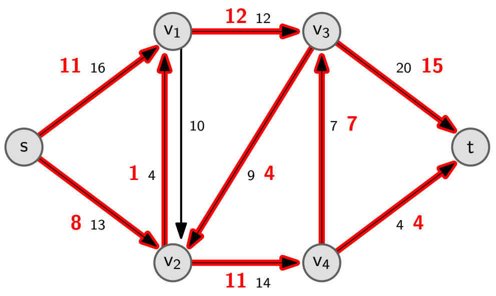


Figure: A flow f (red) in the network $\mathcal{N}$.

Maximum Flows: Introduction

Definition 3 (Value of a Flow, Maximum Flow)

The value of a flow f is the total amount of flow reaching the sink t: $|f| = \sum_{v \in V} f(v,t) = f(V,t)$.

f^* is a maximum flow when there is no flow g with $|g| > |f^*|$.

Definition 4 (Residual Capacity)

Given a flow f in the network $\mathcal{N}$. The residual capacity of an arc $a \in V \times V$ in respect to f is defined as $r_f(a) = \varsigma(a) - f(a)$.

Notice: An arc a with $r_f(a) > 0$ is called a *residual arc* where additional flow can be supplemented; otherwise ($r_f(a) = 0$) the arc is called saturated.

Maximum Flows: Introduction

Definition 5 (Residual Network)

The graph $G_f = (V, A_f)$ with A_f being the set of all residual arcs (i.e., all arcs a with $r_f(a) > 0$) is called the residual network in respect to a given flow f.

Definition 6 (Augmenting Path)

A path P in the residual network G_f from source s to sink t is called an augmenting path in respect to a given flow f.

Notice: An augmenting path in G_f can be used to increase the flow from s to t leading to a new flow f' and residual network $G_{f'}$.

Maximum Flows: Introduction

Definition 7 (Push)

The basic operation of augmenting a flow f along an arc $(u,v) \in A$ by some value x is referred to as a push.

Notice: $f'(u,v) = f(u,v) + x \;\rightarrow\; f'(v,u) = f(v,u) - x$.

Increasing flow f in a network $\mathcal{N}$:

1. Find augmenting path P from s to t in G_f;
   $x \rightarrow$ minimum residual capacity along P.
2. Push x along P (this saturates at least one arc) $\rightarrow f'$.
3. $|f'| = |f| + x$

Maximum Flows: Introduction

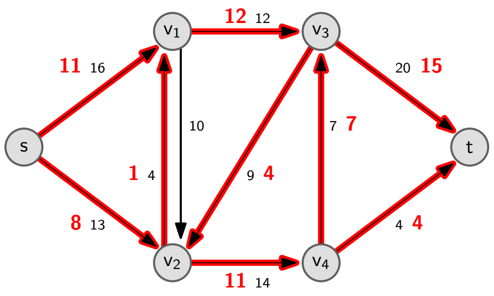


Figure: A flow f (red) in the network $\mathcal{N}$.

Maximum Flows: Introduction

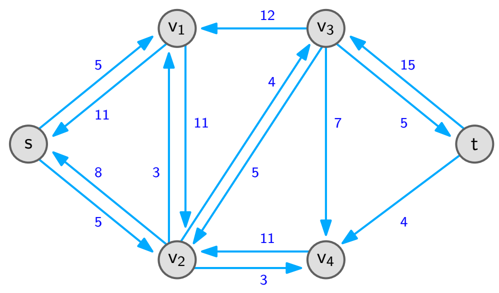


Figure: The residual network G_f in respect to f.

Maximum Flows: Introduction

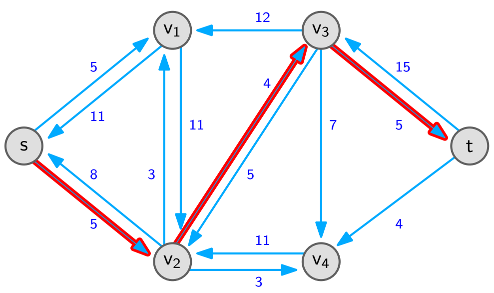


Figure: An augmenting path P in the residual network G_f.

Maximum Flows: Introduction

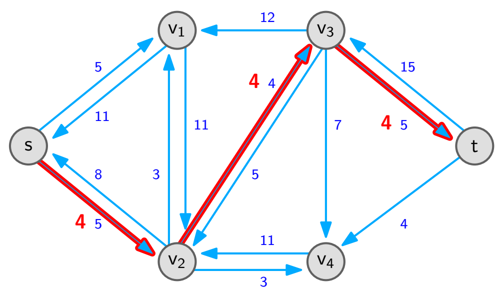


Figure: Minimum residual capacity along P: 4.

Maximum Flows: Introduction

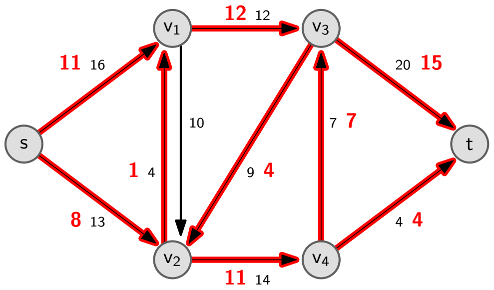


Figure: Flow f before push operation.

Maximum Flows: Introduction

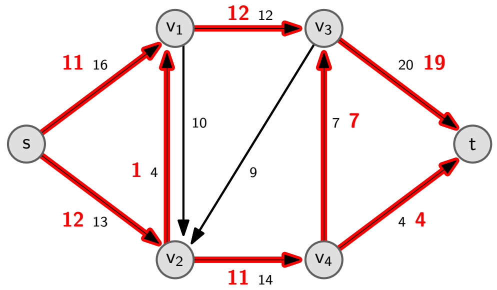


Figure: Flow f' after push operation (4 units along P).

Maximum Flows: Introduction

Question: Flow f $\rightarrow$ augmenting path P in G_f $\rightarrow$ pushes along P $\rightarrow$ new flow f', but is f' really a flow (as defined before)?

Answer: Yes.

Proof:

Skew symmetry:
See push operation: $f(u,v) + x \;\rightarrow\; f(v,u) - x$. ✓

Capacity constraints:
Construction of G_f: Residual capacity $r_f(a)$ gives the max. amount of additional flow the arc a can carry. ✓

Flow conservation:
$\forall u \in V \setminus \{s,t\}$ along P the net flow remains 0:
x is added to the incoming as well as outgoing flow of u. ✓

□

Maximum Flow / Minimum Cut

Definition 8 (Cut, Capacity of a Cut)

A cut is a set of nodes $S \subset V$ with $s \in S$ and $t \in \overline{S}$, where $\overline{S} := V \setminus S$; i.e., a cut is a partition of V into two non-empty sets S and $\overline{S}$.

The capacity of a cut is defined as the capacity of all arcs crossing the cut from S to $\overline{S}$:

$$\varsigma(S,\overline{S}) = \sum_{u \in S} \sum_{v \in \overline{S}} \varsigma(u,v).$$

The number of possible cuts is $2^{|V|-2}$.

Maximum Flow / Minimum Cut

Lemma 9 (Flow / Cut Capacity)

No flow f in a network $\mathcal{N}$ can have a value greater than the capacity of any cut S.

Proof (using the definition of a flow 1–3)**:**

$|f| = f(V,t) \overset{(3)}{=} f(V,t) + f(V, \overline{S} \setminus \{t\}) = f(V, \overline{S}) =$

$f(S, \overline{S}) + f(\overline{S}, \overline{S}) = f(S, \overline{S}) \overset{(2)}{\leq} \varsigma(S, \overline{S})$ $\square$

Implication: $|f| = f(S, \overline{S})$ for any flow f and any cut S in $\mathcal{N}$.

Definition 10 (Saturated Cut)

A flow f saturates a cut S iff $f(S, \overline{S}) = \varsigma(S, \overline{S})$.

Maximum Flow / Minimum Cut

Theorem 11 (Max-Flow / Min-Cut)

Let f be a flow in a network $\mathcal{N}$, then the following three conditions are equivalent:

1. *There is a cut S in $\mathcal{N}$ saturated by f.*
2. *f is a maximum flow in $\mathcal{N}$.*
3. *There is no augmenting path P in the residual network G_f.*

Ford/Fulkerson and Elias/Feinstein/Shannon, 1956

The max-flow min-cut theorem is one of the central statements in optimization theory!

Maximum Flow / Minimum Cut

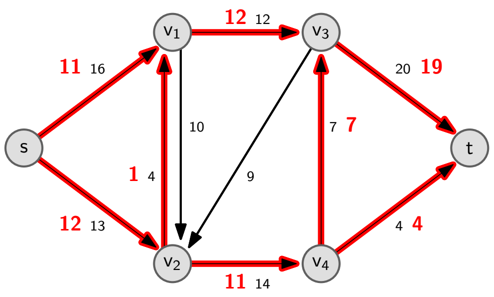


Figure: Flow f' (after a push operation).

Maximum Flow / Minimum Cut

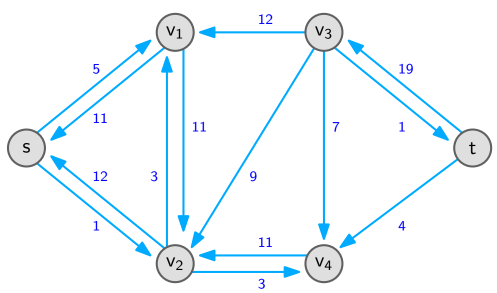


Figure: Residual network $G_{f'}$ in respect to flow f'.

Maximum Flow / Minimum Cut

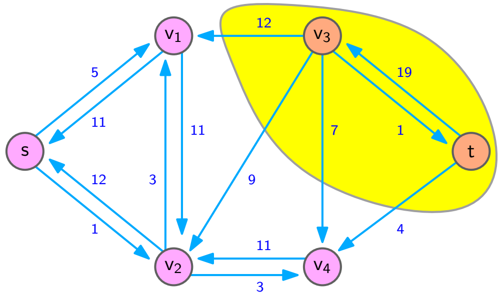


Figure: No augmenting path from s to t in $G_{f'}$.

Maximum Flow / Minimum Cut

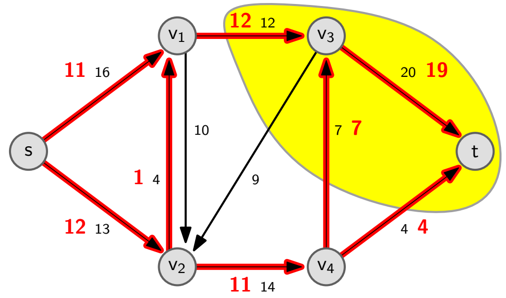


Figure: Maximum flow f^* in network $\mathcal{N}$.

Maximum Flow / Minimum Cut

Proof (max-flow min-cut theorem, circular reasoning)**:**

1→2: saturated cut → maximum flow [based on Lemma 9]
For any flow g and any cut $S \rightarrow |g| \leq \varsigma(S, \overline{S}) \Rightarrow$
f max. flow, because $f(S, \overline{S}) = \varsigma(S, \overline{S})$ due to condition 1.

2→3: maximum flow → no augmenting path
If there would be an augmenting path, f could be increased by some positive value $x \Rightarrow f$ would not be a max. flow.

3→1: no augmenting path → saturated cut
Let S be the set of nodes in G_f reachable from $s \rightarrow s \in S$, and $t \notin S$ due to condition 3 $\rightarrow S$ is a cut $\Rightarrow$
$\forall (u, v) \in A, u \in S, v \in \overline{S} : f(u, v) = \varsigma(u, v)$,
otherwise $r_f(u, v) > 0 \rightarrow (u, v) \in G_f \rightarrow v$ could be reached from $s \rightarrow$ contradiction to definition of S.

□

Maximum Flow / Minimum Cut

Background: Why is it called the *Max-Flow Min-Cut* theorem?

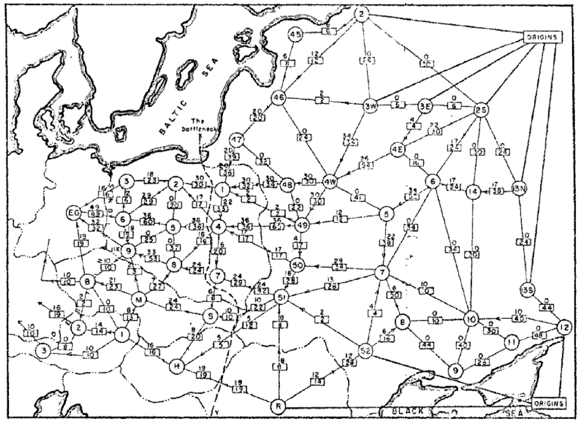
Let f be a maximum flow in $\mathcal{N}$. Lemma 9 states that $|f|$ is bounded above by the capacity of any cut. Therefore a cut S with $\varsigma(S, \overline{S}) = |f|$ has to be a cut of minimum capacity; such a saturated cut exists due to the theorem (2→1).

Suppose there is a cut S' with $\varsigma(S', \overline{S'}) < \varsigma(S, \overline{S})$, then there must hold:

$$|f| \leq \varsigma(S', \overline{S'}) < \varsigma(S, \overline{S}) = |f| \rightarrow \text{contradiction.}$$

Problem became of major interest during cold war between the United States and the Soviet Union from the mid-1940s until the early 1990s: Where to hit the Soviet rail system to prevent transport of troops and supplies to Eastern Europe?

Maximum Flow / Minimum Cut



Harris, Ross: *Fundamentals of a Method for Evaluating Rail Net Capacities*
Research Memorandum RM-1537, 1955

Ford-Fulkerson Algorithm

Ford-Fulkerson Algorithm

- Proposed by Ford and Fulkerson 1962.
- Directly motivated by the proof for the max-flow min-cut theorem: 3 (no augmenting path) $\rightarrow$ 2 (maximum flow).
- Basic idea:
  - Start with a null flow.
  - As long as there can be found an augmenting path:
    Increase flow along this path.
- Ford-Fulkerson algorithm is restricted to integer capacities!

Ford-Fulkerson Algorithm

Procedure `FordFulkerson()`

$i \leftarrow 0$; `/* initialization */`
$f_i \leftarrow$ null flow;

while $\exists$ *augmenting path* P *from* s *to* t *in* G_{f_i} **do** `/* algorithm */`
    $x \leftarrow$ minimum residual capacity along P;
    augment flow of value x along P;
    $f_{i+1} \leftarrow f_i + x$;
    $i \leftarrow i + 1$;

return f_i;

Ford-Fulkerson Algorithm: Example

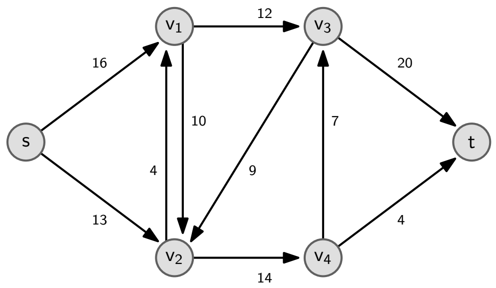


Figure: A flow network $\mathcal{N}$ with associated arc capacities $\varsigma(u, v)$. $\mathcal{N} = G_{f_0}$ since $f(u, v) = 0, \ \forall (u, v) \in A$.

Ford-Fulkerson Algorithm: Example

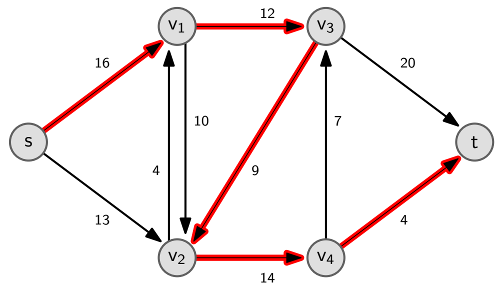


Figure: A first augmenting path in the original network $\mathcal{N}(G_{f_0})$.

Ford-Fulkerson Algorithm: Example

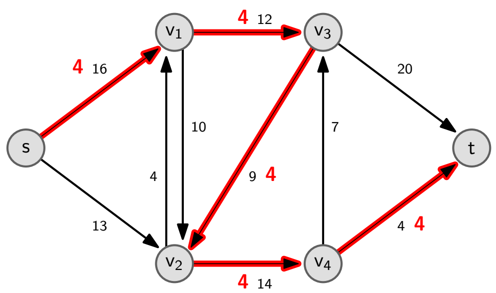


Figure: Minimum capacity along the path: $4 \rightarrow$ flow f_1.

Ford-Fulkerson Algorithm: Example

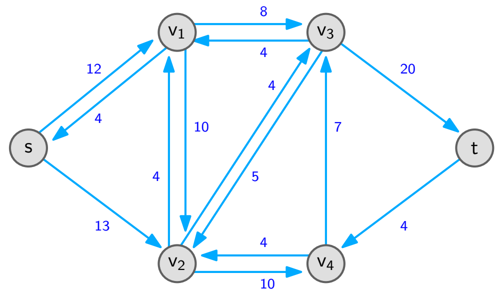


Figure: Residual network G_{f_1}.

Ford-Fulkerson Algorithm: Example

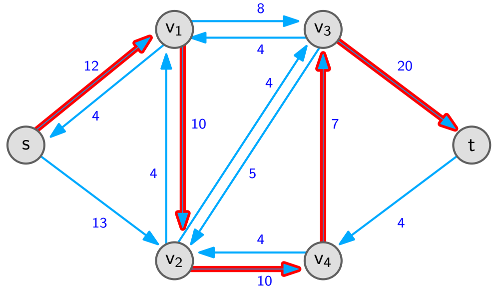


Figure: Augmenting path in G_{f_1}.

Ford-Fulkerson Algorithm: Example

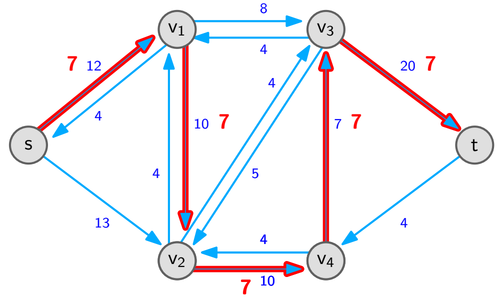


Figure: Minimum capacity along the path: 7.

Ford-Fulkerson Algorithm: Example

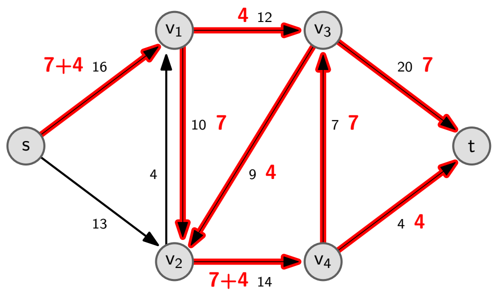


Figure: New flow f_2.

Ford-Fulkerson Algorithm: Example

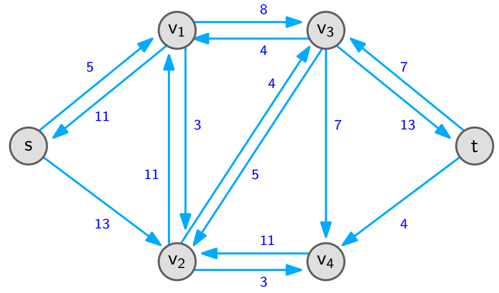


Figure: Residual network G_{f_2}.

Ford-Fulkerson Algorithm: Example

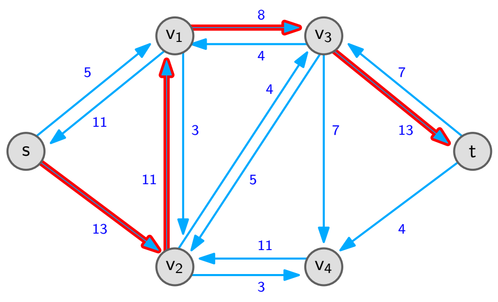


Figure: Augmenting path in G_{f_2}.

Ford-Fulkerson Algorithm: Example

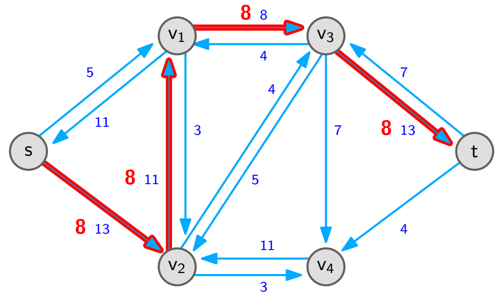


Figure: Minimum capacity along the path: 8.

Ford-Fulkerson Algorithm: Example

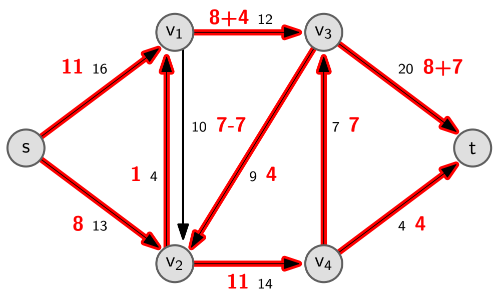


Figure: New flow f_3.

Ford-Fulkerson Algorithm: Example

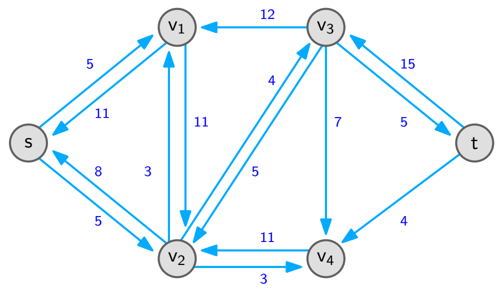


Figure: Residual network G_{f_3}.

Ford-Fulkerson Algorithm: Example

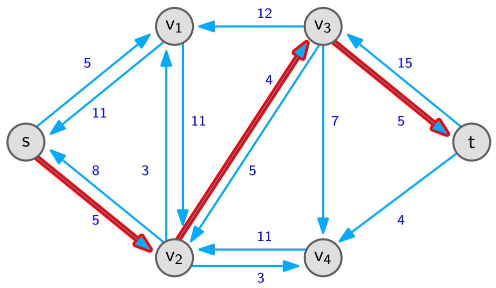


Figure: Augmenting path in G_{f_3}.

Ford-Fulkerson Algorithm: Example

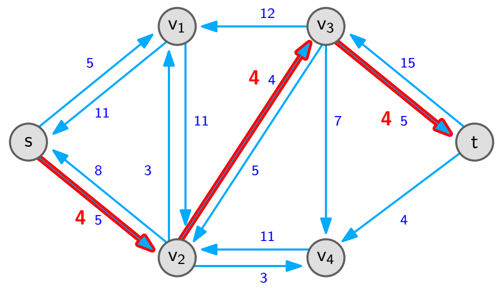


Figure: Minimum capacity along the path: 4.

Ford-Fulkerson Algorithm: Example

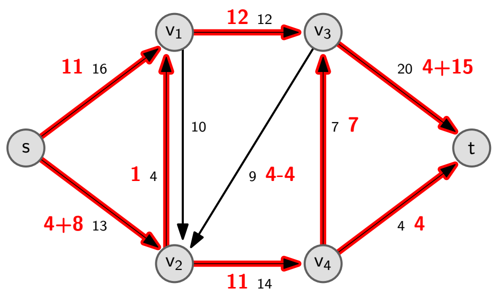


Figure: New flow f_4.

Ford-Fulkerson Algorithm: Example

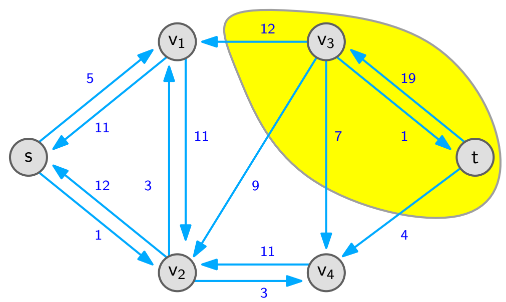


Figure: Residual network G_{f_4}, no augmenting $s-t$ path.

Ford-Fulkerson Algorithm: Example

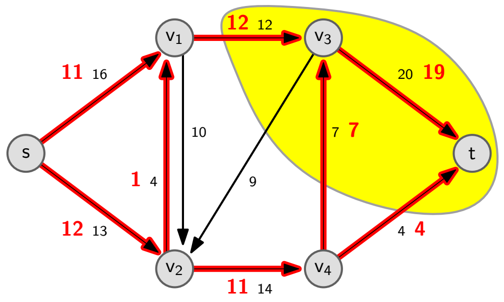


Figure: The maximum flow f_4 in the network $\mathcal{N}$.

Ford-Fulkerson Algorithm

Correctness (without proof)**:** Ford-Fulkerson algorithm terminates computing the maximum flow in case the capacities in the flow network $\mathcal{N}$ are integral.

The maximum flow f^* is integral.

Runtime:

- Null flow: $\Theta(|A|)$.
- Find augmenting path, DFS / BFS: $\Theta(|A|)$.
- Number of augmenting paths: $O(|f^*|)$.
  Integrality condition $\rightarrow$ f is increased at least by 1 in each iteration.

$\Rightarrow$ Ford-Fulkerson algorithm runs in time $O(|A| \cdot |f^*|)$ (worst case).

Ford-Fulkerson Algorithm: Problems

Problem: Pseudopolynomial running time:

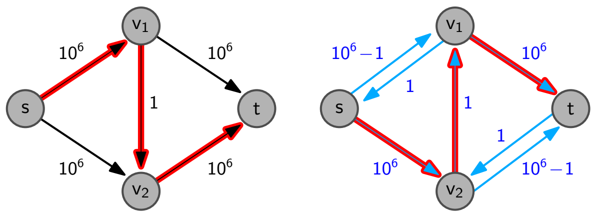
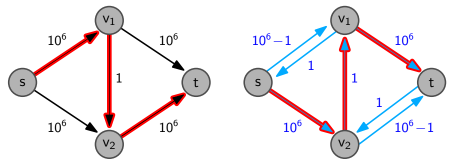


Figure: Flow network $\mathcal{N}$ with $|f^*| = 2 \cdot 10^6$, where the Ford-Fulkerson algorithm requires time $\Theta(|A| \cdot |f^*|)$.

Ford-Fulkerson Algorithm: Problems



Improvement: Edmonds-Karp Algorithm (1972)

Two heuristics for choosing augmenting paths:

- *Fat Pipes:*
  Augmenting path with largest bottleneck value;
  running time $O(|A|^2 \cdot \log|A| \cdot \log|f^*|)$. (modified Prim's MST)
- *Short Pipes:*
  Shortest (in respect to number of arcs) augmenting path;
  running time $O(|V| \cdot |A|^2)$. (BFS)

Ford-Fulkerson Algorithm: Problems

Problem: Capacities $\in \mathbb{R}$, irrational capacities:

- Rational capacities: Scale to integer $\rightarrow$ running time can explode (pseudopolynomial).
- Irrational capacities: Algorithm can loop forever and may converge to a wrong maximum flow value.

Sketch of Proof

Details: Uri Zwick:
The smallest networks on which the Ford-Fulkerson maximum flow procedure may fail to terminate. (1993)

Ford-Fulkerson Algorithm: Problems

Basic idea:
Use a network $\mathcal{N}$ to simulate the computation of a sequence $\langle a_i \rangle$.

$$
\begin{aligned}
\langle a_i \rangle : \quad a_0 &= 1 \\
a_1 &= \phi = \tfrac{\sqrt{5}-1}{2} \approx 0.62 \qquad (\tfrac{\sqrt{5}+1}{2} \ldots \text{golden ratio}) \\
a_i &= a_{i-2} - a_{i-1} \quad \forall i \geq 2 \\
&\vdots \\
a_2 &= a_0 - a_1 = 1 - \phi = \phi^2 \\
a_3 &= \phi - \phi^2 = \phi \cdot (1 - \phi) = \phi \cdot \phi^2 = \phi^3 \\
a_i &= \phi^i
\end{aligned}
$$

Ford-Fulkerson Algorithm: Problems

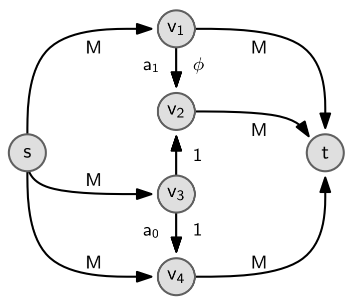
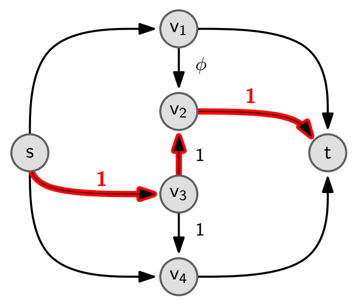


Figure: Flow network $\mathcal{N}$ with irrational capacity $\phi = \frac{\sqrt{5}-1}{2} \approx 0.62$, $M =$ some large integer.

Maximum flow $|f^*| = 2 \cdot M + 1$.

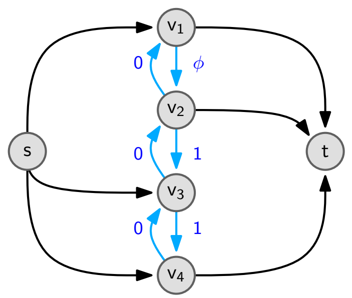
Ford-Fulkerson Algorithm: Problems



$f^{\lightning} = 1$

First flow to "initialize" the network $\mathcal{N}$.

Ford-Fulkerson Algorithm: Problems



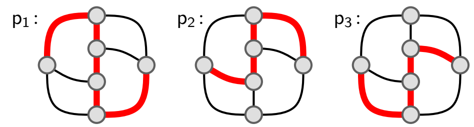
$f^{\lightning} = 1$

The first residual network: $\varsigma(v_3, v_4) = 1 = a_0$, $\varsigma(v_1, v_2) = \phi = a_1$. (only the residual arcs and capacities of the critical edges between the nodes v_i, $i = 1 \dots 4$, are illustrated).

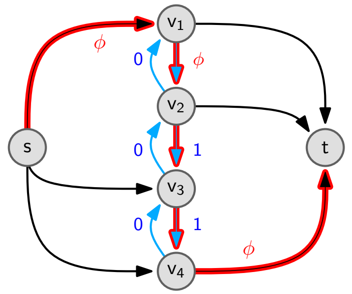
Ford-Fulkerson Algorithm: Problems

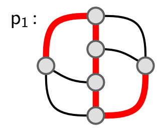
Sequence of augmenting paths p_i, $i = 1 \dots 3$, to simulate the computation of $\langle a_i \rangle$ (infinite loop):

$$p_1 \to p_2 \to p_1 \to p_3 \to \dots$$



Ford-Fulkerson Algorithm: Problems

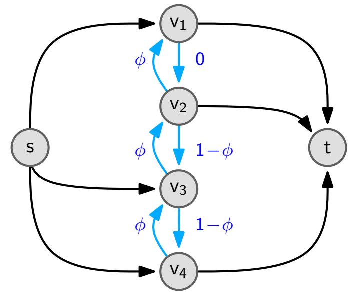




$f^{\lightning} = 1 + \phi$

Flow along augmenting path p_1; bottleneck arc: $v_1 \to v_2$.

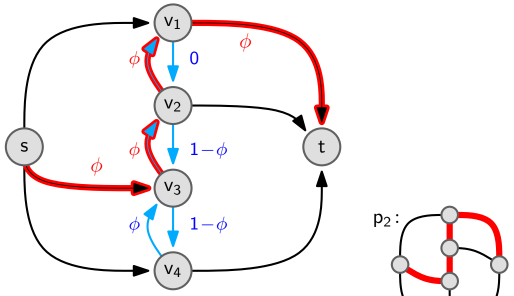
Ford-Fulkerson Algorithm: Problems



$f^{\lightning} = 1 + \phi$

Residual network (capacity of residual arc $v_3 \to v_4 = a_2$).

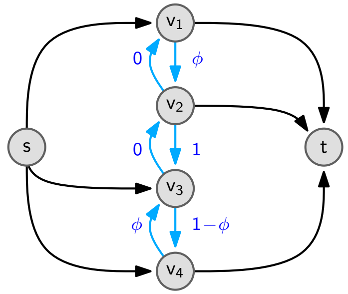
Ford-Fulkerson Algorithm: Problems



$$f^{\lightning} = 1 + \phi + \phi$$

Flow along augmenting path p_2; bottleneck arc: $v_2 \to v_1$.

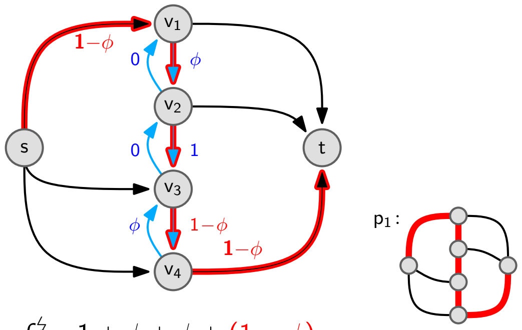
Ford-Fulkerson Algorithm: Problems



$$f^{\lightning} = 1 + \phi + \phi$$

Residual network
(capacities of residual arcs $v_1 \to v_2$ and $v_2 \to v_3$ "resetted").

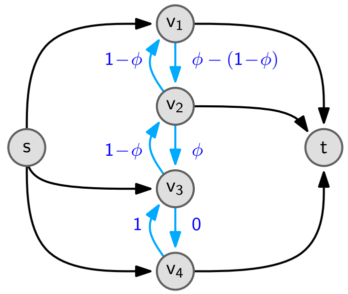
Ford-Fulkerson Algorithm: Problems



$$f^{\lightning} = 1 + \phi + \phi + (1 - \phi)$$

Flow along augmenting path p_1; bottleneck arc: $v_3 \to v_4$.

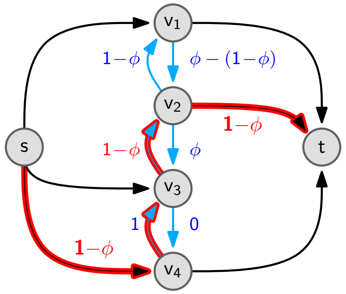
Ford-Fulkerson Algorithm: Problems

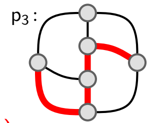


$$f^{\lightning} = 1 + \phi + \phi + (1 - \phi)$$

Residual network
(capacity of residual arc $v_1 \to v_2 = a_3$).

Ford-Fulkerson Algorithm: Problems

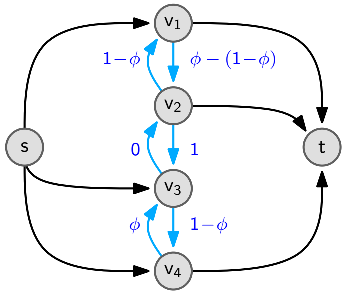




$$f^{\lightning} = 1+\phi+\phi+(1-\phi)+(1-\phi)$$

Flow along augmenting path p_3; bottleneck arc: $v_3 \to v_2$.

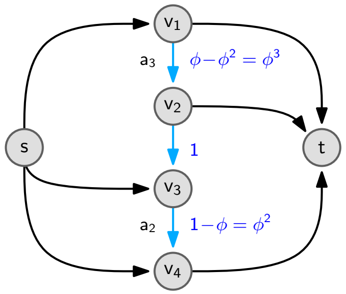
Ford-Fulkerson Algorithm: Problems



$$f^{\lightning} = 1+\phi+\phi+(1-\phi)+(1-\phi)$$

Residual network
(capacity of residual arc $v_3 \to v_4$ "resetted").

Ford-Fulkerson Algorithm: Problems



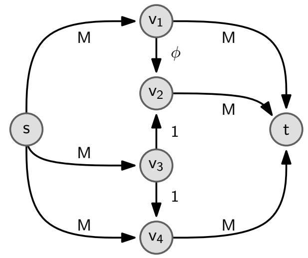
$$f^{\lightning} = 1+\phi+\phi+(1-\phi)+(1-\phi)$$

Residual network
($p_1 \to p_2 \to p_1 \to p_3$: $a_1 \to a_3$ and $a_0 \to a_2$).

Ford-Fulkerson Algorithm: Problems

$$\begin{aligned} f^{\lightning} &= 1+\phi+\phi+(1-\phi)+(1-\phi)+\ldots = \\ &= 1+2\cdot\phi+2\cdot\phi^2+2\cdot\phi^3+\ldots = \\ &= 1+2\cdot\sum_{i=1}^{\infty}\phi^i = \text{(geometric series)} \\ &= 1+2\cdot\left(\frac{1}{1-\phi}\right)-2 = \\ &= 2+\sqrt{5} < \mathbf{5}. \end{aligned}$$

Remember:
Maximum flow $|f^*|$ in network $\mathcal{N}$: $2\cdot M+1$.



Ford-Fulkerson Algorithm: Problems

Drawback of all augmenting path algorithms:

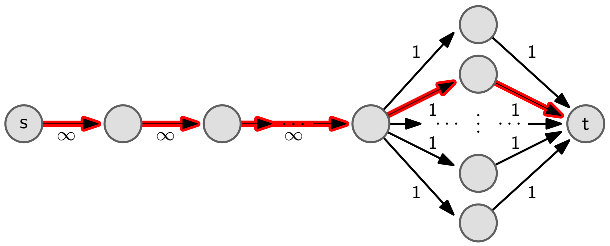


Figure: Sending flow along a $s-t$ path is a computationally expensive operation, it requires $O(n)$ time in worst case.

Improvement: Preflow-Push algorithms.

Preflow-Push Algorithm

Generic Preflow-Push Algorithm

- Proposed by Goldberg and Tarjan 1988.
- Running time: $O(|V|^2 \cdot |A|)$.
  For comparison, the Edmonds-Karp algorithms:
  $O(|A|^2 \cdot \log |A| \cdot \log |f^*|)$ resp. $O(|V| \cdot |A|^2)$.
- Basic idea:
  - Relax flow conservation rule.
  - Push flow along individual arcs, not along complete $s-t$ paths.
  - Every node has an "overfall basin" of unlimited size to buffer flow.
  - Direct flow from basins with excess to the target t.
- Preflow-Push algorithm is not restricted to integer capacities!

Generic Preflow-Push Algorithm

Definition 12 (Preflow)

A preflow is a real function $f: V \times V \to \mathbb{R}$ with the following three properties:

1. Skew symmetry: $f(u,v) = -f(v,u) \quad \forall u,v \in V$
2. Capacity constraints: $f(u,v) \leq \varsigma(u,v) \quad \forall u,v \in V$
3. Excess condition: $f(V,u) = e_f(u) \geq 0 \quad \forall u \in V \setminus \{s\}$

Intermediate stages:

- Augmenting path algorithms $\to$ feasible flows.
- Preflow-push algorithms $\to$ infeasible flows (preflows).

Generic Preflow-Push Algorithm

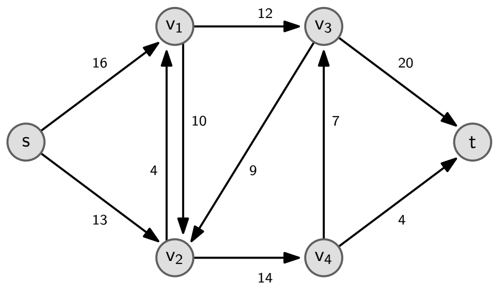


Figure: Illustration of the preflow-push algorithm (basic idea, no labels).

Maximum Flow Basics Ford-Fulkerson Preflow-Push Lower Bounds Minimum Cost Flow

Generic Preflow-Push Algorithm

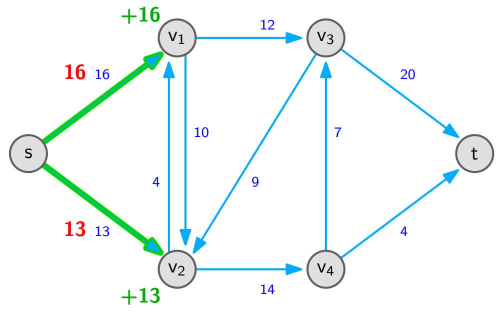


Figure: Initialization: Saturate all arcs having s as their source node; $e_f(v_1) = 16$, $e_f(v_2) = 13$.

Maximum Flow Basics Ford-Fulkerson Preflow-Push Lower Bounds Minimum Cost Flow

Generic Preflow-Push Algorithm

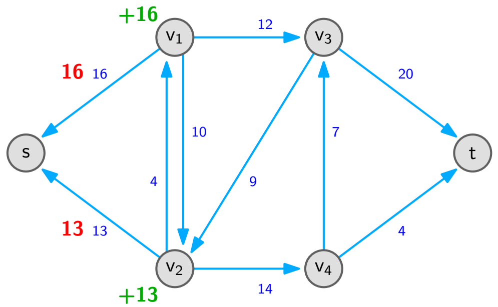


Figure: Residual network G_f; two nodes $\neq t$ with excess $\rightarrow$ continue.

Maximum Flow Basics Ford-Fulkerson Preflow-Push Lower Bounds Minimum Cost Flow

Generic Preflow-Push Algorithm

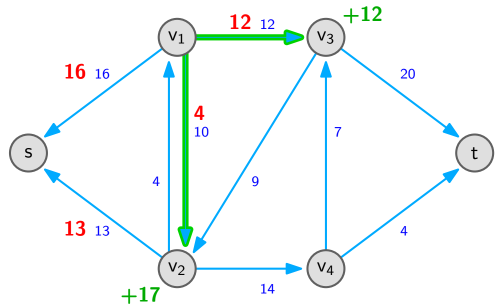


Figure: Push excess of v_1 ($e_f(v_1) = 16$) to nodes v_2 and v_3; $e_f(v_2) = 13 + 4$, $e_f(v_3) = 12$.

Generic Preflow-Push Algorithm

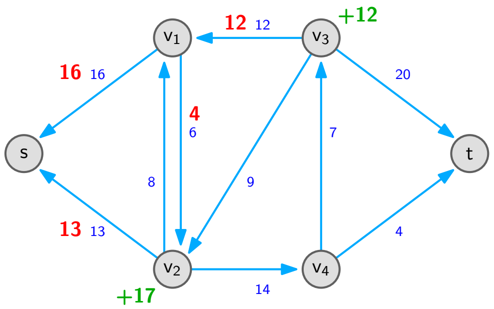


Figure: Residual network G_f; two nodes $\neq t$ with excess $\rightarrow$ continue.

Maximum Flow Basics Ford-Fulkerson Preflow-Push Lower Bounds Minimum Cost Flow

Generic Preflow-Push Algorithm

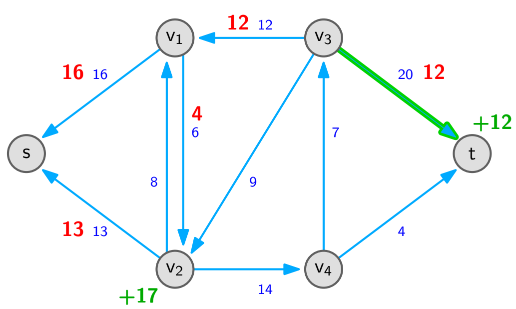


Figure: Push excess of v_3 ($e_f(v_3) = 12$) to the target node t;
$e_f(v_2) = 17$, $e_f(t) = \mathbf{12}$.

Maximum Flow Basics Ford-Fulkerson Preflow-Push Lower Bounds Minimum Cost Flow

Generic Preflow-Push Algorithm

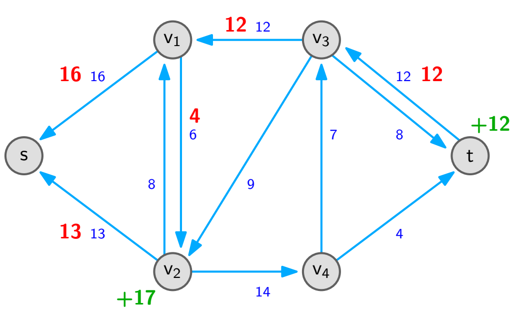


Figure: Residual network G_f; one node $\neq t$ with excess $\rightarrow$ continue.

Maximum Flow Basics Ford-Fulkerson Preflow-Push Lower Bounds Minimum Cost Flow

Generic Preflow-Push Algorithm

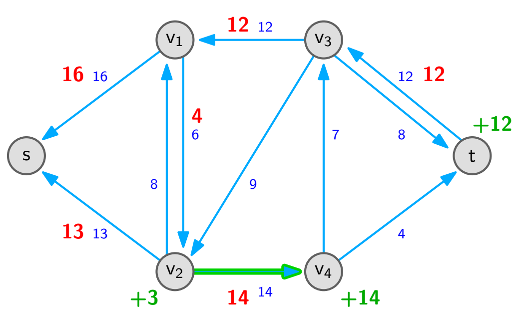


Figure: Push as much as possible excess of v_2 ($e_f(v_2) = 17$) to node v_4;
$e_f(v_2) = 17 - 14$, $e_f(v_4) = 14$, $e_f(t) = \mathbf{12}$.

Generic Preflow-Push Algorithm

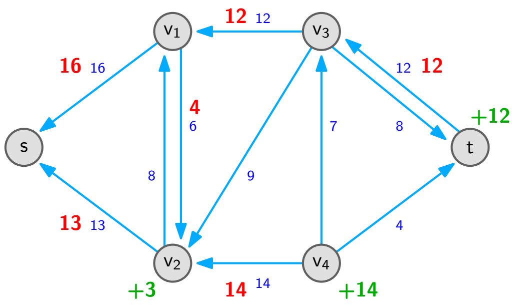


Figure: Residual network G_f; two nodes $\neq t$ with excess $\rightarrow$ continue.

Generic Preflow-Push Algorithm

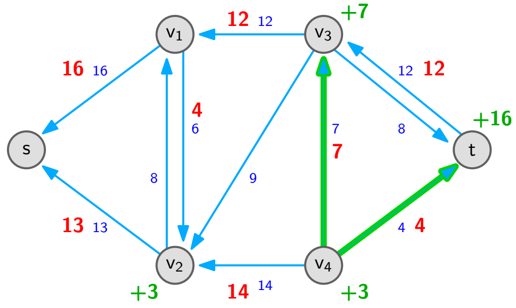


Figure: Push as much as possible excess of v_4 ($e_f(v_4) = 14$) to v_3 and t; $e_f(v_2) = 3$, $e_f(v_3) = 7$, $e_f(v_4) = 14 - 11$, $e_f(t) = \mathbf{12} + \mathbf{4}$.

Generic Preflow-Push Algorithm

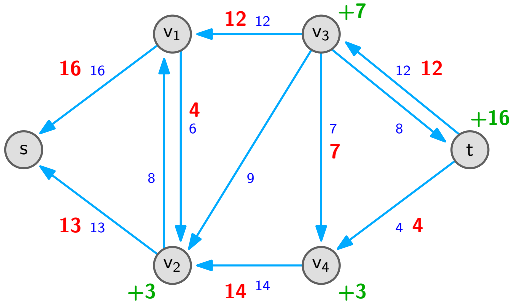


Figure: Residual network G_f; three nodes $\neq t$ with excess $\rightarrow$ continue.

Generic Preflow-Push Algorithm

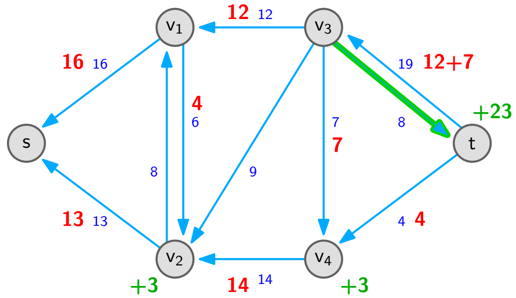


Figure: Push excess of v_3 ($e_f(v_3) = 7$) to the target node t; $e_f(v_2) = 3$, $e_f(v_4) = 3$, $e_f(t) = \mathbf{16} + \mathbf{7}$.

Generic Preflow-Push Algorithm

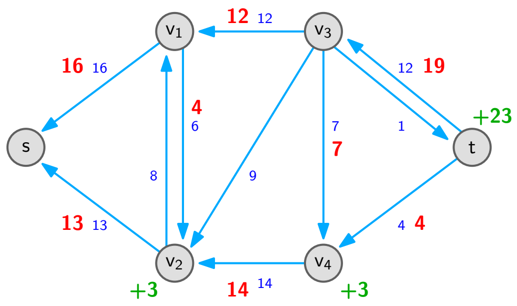


Figure: Residual network G_f; two nodes $\neq t$ with excess $\rightarrow$ continue.

Maximum Flow Basics Ford-Fulkerson Preflow-Push Lower Bounds Minimum Cost Flow

Generic Preflow-Push Algorithm

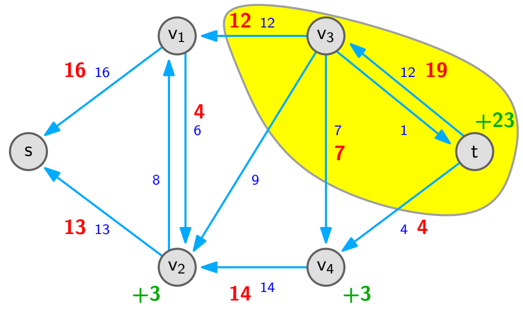


Figure: No possibility to push excess of nodes v_2 and v_4 to target $t \rightarrow$ send excess back to source s.

Maximum Flow Basics Ford-Fulkerson Preflow-Push Lower Bounds Minimum Cost Flow

Generic Preflow-Push Algorithm

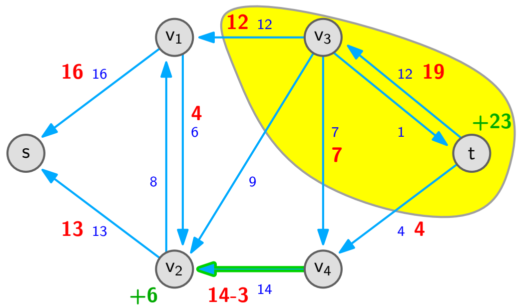


Figure: Push excess of v_4 ($e_f(v_4) = 3$) back to v_2; $e_f(v_2) = 3 + 3$, $e_f(t) = \mathbf{23}$.

Maximum Flow Basics Ford-Fulkerson Preflow-Push Lower Bounds Minimum Cost Flow

Generic Preflow-Push Algorithm

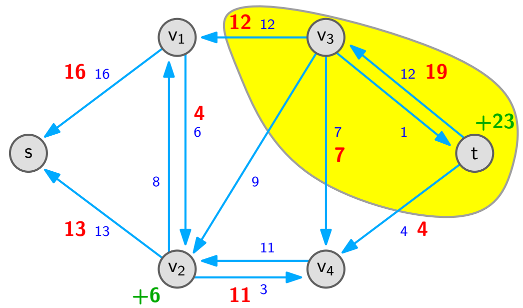


Figure: Residual network G_f; one node $\neq t$ with excess $\rightarrow$ continue.

Generic Preflow-Push Algorithm

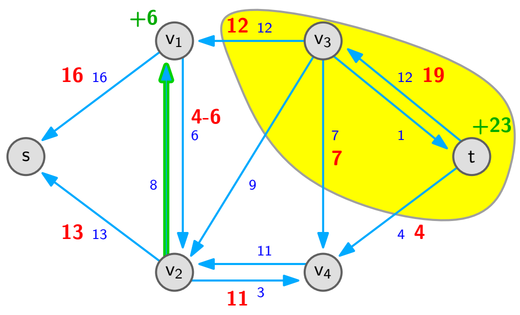


Figure: Push excess of v_2 ($e_f(v_2) = 6$) back to v_1; $e_f(v_1) = 6$, $e_f(t) = \mathbf{23}$.

Generic Preflow-Push Algorithm

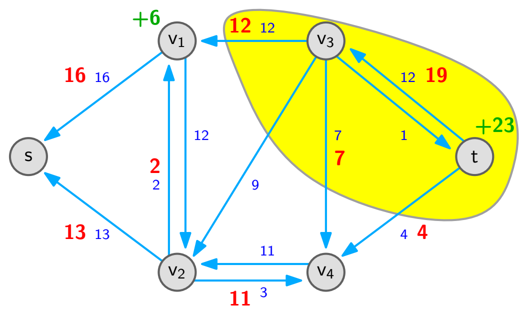


Figure: Residual network G_f; one node $\neq t$ with excess $\rightarrow$ continue.

Generic Preflow-Push Algorithm

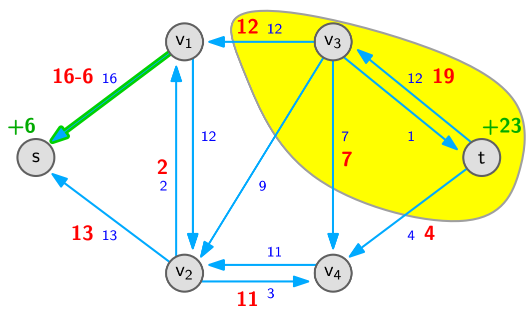


Figure: Push excess of v_1 ($e_f(v_1) = 6$) back to source node s; $e_f(t) = \mathbf{23}$.

Generic Preflow-Push Algorithm

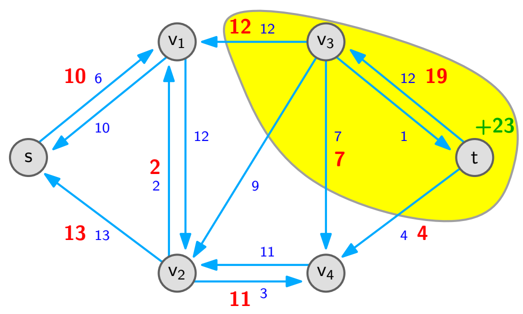


Figure: Residual network G_f; no node $\neq t$ with excess $\rightarrow$ terminate.

Generic Preflow-Push Algorithm

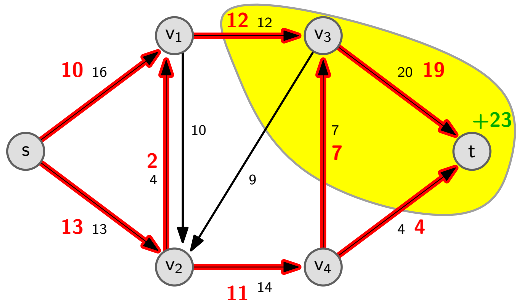


Figure: Valid maximum flow within network $\mathcal{N}$: $|f^*| = e_f(t) = 23$.

Generic Preflow-Push Algorithm

Definition 13 (Label / Height of a Node; Valid Labeling)

Label / Height: Function $d : V \to \mathbb{N}_0$.

A labeling is called valid:

- $d(s) = |V| = n$ and $d(t) = 0$
- $d(u) \leq d(v) + 1 \quad \forall$ residual arcs (u, v) in G_f

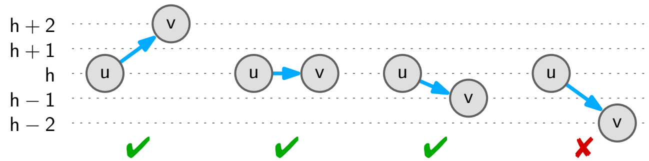


Figure: Valid and invalid labeling of nodes u and v in a residual network G_f; declining arcs are only allowed if the difference in height is not more than 1.

Generic Preflow-Push Algorithm

Definition 14 (Admissible Arc)

An arc (u, v) in G_f (i.e., $f(u, v) < \varsigma(u, v)$) is called admissible iff $d(u) = d(v) + 1$ (declining arc).

Definition 15 (Active Node)

A node $v \in V \setminus \{s, t\}$ is called active iff the excess $e_f(v) > 0$.

Definition 16 (Saturating / Nonsaturating Push)

Let u be an active node.
A saturating push of value x along a residual arc (u, v) in G_f removes this arc from the residual network ($x = r_f(u, v)$).
A nonsaturating push along (u, v) reduces excess at u to zero ($e_f(u) = x < r_f(u, v)$).

Generic Preflow-Push Algorithm

Procedure push(u,v)

/* precondition: u active, (u, v) admissible */

$x \leftarrow \min\{r_f(u, v), e_f(u)\}$;
$f(u, v) \leftarrow f(u, v) + x$;
$f(v, u) \leftarrow -f(u, v)$;

Procedure lift(u)

/* precondition: u active, no admissible arc (u, v) */

$d(u) \leftarrow d(u) + 1$;

Generic Preflow-Push Algorithm

Procedure `GenericPreflowPush(u,v)`

$d(s) \leftarrow n$;   `/* initialization */`
forall $v \in V \setminus \{s\}$ **do** $d(v) \leftarrow 0$;
forall $(u, v) \in A$ **do** $f(u, v) = f(v, u) \leftarrow 0$;
forall $(s, v) \in A$ **do**
    $f(s, v) \leftarrow \varsigma(s, v)$;
    $f(v, s) \leftarrow -f(s, v)$;

while $\exists$ *active node* $u \in G_f$ **do**   `/* algorithm */`
    if $\exists$ *admissible arc* $(u, v) \in G_f$ **then**
        push(u, v);
    else
        lift(u);

Generic Preflow-Push Algorithm

Lemma 17 (Labeling / Preflow)

The labeling d is always valid and f is always a preflow.

Proof:

Initialization:

- Preflow: Flow f is a preflow. ✓
- Labeling: Labeling d is valid because of saturation of arcs (s, v). ✓

`lift(u)`:

- Preflow: f is not modified by a `lift()` operation. ✓
- Labeling: preconditions of `lift()` operation: $\forall (u, v) \in G_f$: $d(u) \le d(v)$, otherwise `push()` would have been called $\Rightarrow d(u) + 1$ cannot lead to an invalid labeling. ✓

Generic Preflow-Push Algorithm

Lemma 17 (Labeling / Preflow)

The labeling d is always valid and f is always a preflow.

Proof:

`push(u,v)`:

- Preflow: Skew symmetry, capacity constraints, excess condition. ✓
- Labeling: Perhaps arc $(v, u) \in G_f$ after `push()` operation; precondition: $d(u) = d(v) + 1 \Rightarrow d(v) \le d(u) + 1 \Rightarrow$ valid labeling. ✓

□

Generic Preflow-Push Algorithm

Lemma 18 (No Augmenting Path)

Let d be a valid labeling, and f a preflow: There exists no augmenting path from s to t in G_f.

Proof:

An augmenting path from s to t cannot consist of more than $n - 1$ ($|V| = n$) arcs. Due to the definition of a valid labeling $d(s) = n$, $d(t) = 0$, and there exists no arc $(u, v) \in G_f$ with $d(u) > d(v) + 1$.

With a valid labeling it is not possible to connect a node at height n with a node at height 0 without "skipping" at least one level if the path consists of only $n - 1$ arcs.

□

Generic Preflow-Push Algorithm

Lemma 19 (Partial Correctness of Preflow Push Algorithm)

In case the generic preflow-push algorithm terminates f is a maximum flow in the network $\mathcal{N}$.

Proof:

Algorithm terminates $\rightarrow$

- no active nodes, i.e., $e_f(v) = 0 \quad \forall v \in V \setminus \{s,t\} \rightarrow$
- f is not only a preflow but a flow;
- according to Lemma 18 there exists no augmenting $s-t$ path in G_f, f is a valid flow $\Rightarrow$
- f is a maximum flow. □

Still to prove: Algorithm terminates. $\Rightarrow$ Worst-case runtime?

Generic Preflow-Push Algorithm

Lemma 20 (Excess Nodes Connected To Source)

Let f be a preflow and u an active node, i.e., $e_f(u) > 0$:
There exists a path from u to source s in the residual graph G_f.

Proof:

Let $T \subseteq V$ be the set of nodes reachable from u in G_f, and $\overline{T} = V \setminus T$, then the following holds:

$$\sum_{v \in T} e_f(v) = f(V,T) = f(T,T) + f(\overline{T},T) \overset{(1)}{=} f(\overline{T},T) \le 0.$$

$f(\overline{T},T)$ cannot be positive: A flow $f(w,v) > 0$ from a node $w \in \overline{T}$ to a node $v \in T$ would lead to a residual arc (v,w) in $G_f \rightarrow$ contradiction to the definition of T (v is reachable from u, but an arc (v,w) would make node w also reachable from $u \Rightarrow w \in T$ and $w \in \overline{T} \rightarrow ↯$).

Generic Preflow-Push Algorithm

Lemma 20 (Excess Nodes Connected To Source)

Let f be a preflow and u an active node, i.e., $e_f(u) > 0$:
There exists a path from u to source s in the residual graph G_f.

Proof:

Let $T \subseteq V$ be the set of nodes reachable from u in G_f, and $\overline{T} = V \setminus T$, then the following holds:

$$\sum_{v \in T} e_f(v) = f(V,T) = f(T,T) + f(\overline{T},T) \overset{(1)}{=} f(\overline{T},T) \le 0.$$

Excess condition (preflow definition): $e_f(v) \ge 0 \quad \forall v \in V \setminus \{s\}$, and u is an active node ($e_f(u) > 0$) $\Rightarrow$ there has to be a negative term in the sum above $\Rightarrow$ the source node s has to be element of set T and is therefore reachable from u. □

Generic Preflow-Push Algorithm

Lemma 21 (Height Restriction)

For every node $u \in V$: $d(u) \le 2 \cdot n - 1$.

Proof:

It is sufficient to prove this for active nodes, because inactive nodes are not "lifted":

- Lemma 20: There is a path P from u to s in the residual graph G_f,
- which cannot consist of more than $n-1$ arcs;
- $d(s) = n$ and a valid labeling $\Rightarrow$
- $d(s) + n - 1$ is an upper bound for the height of u, i.e., $d(u) \le 2 \cdot n - 1$.

□

Generic Preflow-Push Algorithm

Lemma 22 (Number of Relabel Operations)

The number of relabel resp. `lift()` *operations is bounded above by* $2 \cdot n^2$.

Proof:

Direct consequence of Lemma 21:

- `lift()` increases the height of a node by 1,
- no operation decreases the height of a node,
- $n - 2$ nodes (without s, t),
- $2n - 1$ is an upper bound for the height of each node

$\Rightarrow$ a maximum of $2n^2 - 5n + 2 \leq 2n^2$ `lift()` operations can be performed.

□

Generic Preflow-Push Algorithm

Lemma 23 (Number of Saturating Pushes)

The number of saturating `push()` *operations is bounded above by* $2 \cdot n \cdot m$ $(m = |A|)$.

Proof:

For two consecutive saturating pushes along an arc $(u, v) \in A$ the heights of the nodes u and v have to increase at least by 2.

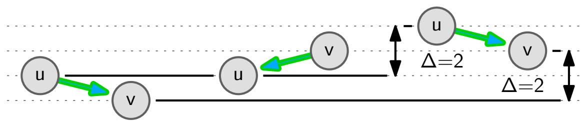


Figure: Saturating push along (u, v) removes this arc from G_f; to perform another saturating push along it there has to be a push along (v, u) to bring back (u, v) into G_f.

Generic Preflow-Push Algorithm

Lemma 23 (Number of Saturating Pushes)

The number of saturating `push()` *operations is bounded above by* $2 \cdot n \cdot m$ $(m = |A|)$.

Proof:

For two consecutive saturating pushes along an arc $(u, v) \in A$ the heights of the nodes u and v have to increase at least by 2.

- $2 \cdot n - 1$ is an upper bound for the height of the nodes u and $v \Rightarrow$ arc (u, v) can be saturated maximum n times,
- the number of arcs in G_f can be up to $2 \cdot m$ (for every arc $(u, v) \in A$ there can also be an arc $(v, u) \in G_f$)

$\Rightarrow$ number of saturating pushes is bounded above by $n \cdot 2 \cdot m$. □

Generic Preflow-Push Algorithm

Lemma 24 (Number of Nonsaturating Pushes)

The number of nonsaturating `push()` *operations is* $\leq 6 \cdot n^2 \cdot m$.

Proof:

Let X be the – changing over time – set of active nodes. We define the following potential function:

$$\Phi = \sum_{u \in X} d(u)$$

At the beginning $\Phi = 0$, and during execution of the algorithm $\Phi \geq 0$.

Generic Preflow-Push Algorithm

Lemma 24 (Number of Nonsaturating Pushes)

The number of nonsaturating `push()` *operations is* $\leq 6 \cdot n^2 \cdot m$.

Proof:

A nonsaturating push along arc (u, v) reduces the excess at u to $0 \rightarrow X = X \setminus \{u\}$. Let Φ' be the resulting potential.

$$\Phi' = \begin{cases} \Phi - d(u) & \text{if } v \text{ was already} \in X, \text{or } v = t, \\ \Phi - d(u) + d(v) & \text{if } v \text{ was} \notin X, \end{cases}$$

$$\Rightarrow \Phi' \leq \Phi - d(u) + d(v) = \Phi - 1,$$

because (u, v) has to be an admissible arc $\rightarrow d(v) = d(u) - 1$.

Generic Preflow-Push Algorithm

Lemma 24 (Number of Nonsaturating Pushes)

The number of nonsaturating `push()` *operations is* $\leq 6 \cdot n^2 \cdot m$.

Proof:

Saturating pushes:

- Push along (u, v) can insert v into set X,
- $d(v) \leq 2 \cdot n - 1$ (Lemma 21),
- number of saturating pushes $= 2 \cdot n \cdot m$ (Lemma 23)

$\Rightarrow$ saturating pushes can increase Φ at most by $4 \cdot n^2 \cdot m$.

Generic Preflow-Push Algorithm

Lemma 24 (Number of Nonsaturating Pushes)

The number of nonsaturating `push()` *operations is* $\leq 6 \cdot n^2 \cdot m$.

Proof:

`lift()` operations:

- `lift(u)` increases $d(u)$ by 1,
- number of `lift()` operations is bounded by $2 \cdot n^2$ (Lemma 22) $\Rightarrow$
- `lift()` operations can increase Φ at most by $2 \cdot n^2$.

$\Rightarrow$ The number of nonsaturating pushes is bounded above by

$$4 \cdot n^2 \cdot m + 2 \cdot n^2 \leq 6 \cdot n^2 \cdot m.$$

□

Generic Preflow-Push Algorithm

Theorem 25 (Correctness of Preflow Push Algorithm)

The generic preflow-push algorithm terminates after $O(n^2 \cdot m)$ `push()` *and* `lift()` *operations, and calculates the maximum flow* f *in the network* $\mathcal{N}$.

Proof:

Direct consequence of the Lemmas 17 to 24.

□

Note: This theorem also proves that every network $\mathcal{N} = (V, A, \varsigma, s, t)$ has a maximum flow.

Preflow-Push Algorithm: Improvements

Improvements **without changing** the worst case runtime complexity:

Definition 26 (Maximum Preflow)

A preflow with the maximum possible flow into target node t is called a maximum preflow.

Definition 27 ($V^{\not\to}$)

$V^{\not\to} \subset V$ is the set of nodes with no directed path to t in the residual network G_f (nodes disconnected from sink).

After initialization $V^{\not\to} = \{s\}$.

Preflow-Push Algorithm: Improvements

Improvements **without changing** the worst case runtime complexity:

Generic preflow-push algorithm performs `push()` and `lift()` operations at active nodes until

1. all excess reaches target node t, or
2. excess returns to the source node s.

- Maximum preflow established $\rightarrow$
- push excess of active nodes back to s (to transform preflow into a flow) $\rightarrow$
- a substantially large number of subsequent `push()`/`lift()` operations is required to raise these nodes until they are sufficiently higher than n.

Preflow-Push Algorithm: Improvements

Improvements **without changing** the worst case runtime complexity:

Improvement 1:

- Start with set $V^{\not\to} = \{s\}$.
- $V^{\not\to} \cup u$, if $d(u) \geq n$.
- Perform no `push()`/`lift()` on nodes $\in V^{\not\to}$.
- Stop algorithm when there are no active nodes in $V \setminus V^{\not\to}$.

At termination, the current preflow is also an optimal preflow $\rightarrow$ convert maximum preflow into maximum flow [exercise] $\rightarrow$ substantial reduce in running time due to empirical tests.

Preflow-Push Algorithm: Improvements

Improvements **without changing** the worst case runtime complexity:

Improvement 2:

- Start with set $V^{\not\to} = \{s\}$.
- Occasionally perform reverse BFS from t in G_f to
  - obtain exact labels / heights, and to
  - add all nodes not reachable from t to $V^{\not\to}$.
- Perform no `push()`/`lift()` on nodes $\in V^{\not\to}$.
- Stop algorithm when there are no active nodes in $V \setminus V^{\not\to}$.

"Occasionally": After $\alpha \cdot n$ `lift()` operations (α constant) $\rightarrow$ does not change the worst case complexity [exercise].

Preflow-Push Algorithm: Improvements

Improvements **changing** the worst case runtime complexity:

Bottleneck: Number of nonsaturating pushes.
Different rules to select active nodes → various algorithms that can substantially reduce these bottleneck operations.

Definition 28 (Node Examination)

Whenever an active node u is selected by the algorithm, it keeps pushing flow from that node until

- the excess of u becomes 0 (saturating pushes except the last one which could be a nonsaturating push), or
- u is lifted.

This sequence of operations is referred to as node examination.

Preflow-Push Algorithm: Improvements

Improvements **changing** the worst case runtime complexity:

FIFO Preflow-Push Algorithm:

- All active nodes are stored in a queue Q.
- Get node u from Q, examine u:
  - Add new active nodes to rear of Q;
  - if u is lifted (still excess available) → add u to rear of Q and continue with next node in Q;
  - if u becomes inactive → continue with next node in Q.
- Stop algorithm when queue of active nodes is empty.

Worst case running time: $O(n^3)$.

Preflow-Push Algorithm: Improvements

Improvements **changing** the worst case runtime complexity:

Highest-Label Preflow-Push Algorithm:

- Push flow from an active node u with highest distance label $d(u)$.

How to select a node with highest $d(\cdot)$ without too much effort?

- `active[k]`, $k = 0, \ldots, 2 \cdot n - 1$: list of active nodes with $d(\cdot) = k$.
- `level`: highest value of k where `active[k]` is nonempty:
  - `lift(u)` of an examined node u → `level` = `level` + 1
  - `active[k]` gets empty without `lift()` operation → check `active[k-1]`, `active[k-2]`, ..., until nonempty list found; total increase in level bounded by $2 \cdot n^2$ (max. number of `lift()` operations) → decrease = $O(2 \cdot n^2)$.

Worst case: $O(n^2 \cdot \sqrt{m})$, currently most efficient method in practice.

Maximum Flow: Networks with Lower Capacity Bounds

Networks with Lower Capacity Bounds

Definition 29 (Flow Network with Lower and Upper Bounds)

A flow network with lower and upper capacity bounds is a 6-tuple $\mathcal{N} = (V, A, \varsigma^L, \varsigma^U, s, t)$, with (V, A) being a directed graph with node set V and arc set A, two nodes $s, t \in V, s \neq t$, and two functions $\varsigma^L, \varsigma^U : A \to \mathbb{R}_{\geq 0}$, the lower ($\varsigma^L$) and upper ($\varsigma^U$) capacity bounds, respectively.

It must hold: $\varsigma^L(a) \leq \varsigma^U(a) \qquad \forall$ arcs $a \in A$.

Extension: $\varsigma^L(a) = \varsigma^U(a) = 0 \qquad \forall$ arcs $a \in (V \times V) \setminus A$.

Networks with Lower Capacity Bounds

Definition 30 (Flow with Nonnegative Lower Bounds)

A flow is a real function $f : V \times V \to \mathbb{R}$ with the following two properties:

1. Capacity constraints: $\varsigma^L(u,v) \leq f(u,v) \leq \varsigma^U(u,v) \qquad \forall u, v \in V$
2. Flow conservation: $f(V,u) - f(u,V) = 0 \qquad \forall u \in V \setminus \{s,t\}$

Note: Skew symmetry has to be discarded.

Networks with Lower Capacity Bounds

Problem: No guarantee that there is a feasible solution to the maximum flow problem in an arbitrary network $\mathcal{N}$ with nonnegative lower and upper bounds:

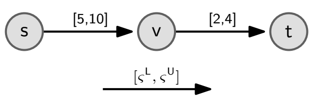


Figure: A flow network $\mathcal{N}$ with no feasible solution.

Two-phase approach to solve the maximum flow problem:

1. Determine whether the problem is feasible, and if so
2. compute a maximum flow in a transformed network $\mathcal{N}'$ without lower bounds.

Networks with Lower Capacity Bounds

Phase 2 (Maximum Flow):

Precondition:
f is a feasible flow, in particular: $\varsigma^L(a) \leq f(a) \leq \varsigma^U(a) \quad \forall a \in (V \times V)$.

Build residual graph G_f using the following residual capacities:

$$r_f(u,v) = \Big(\varsigma^U(u,v) - f(u,v)\Big) + \Big(f(v,u) - \varsigma^L(v,u)\Big)$$

Note: $r_f(u,v)$ is always nonnegative.

- Compute maximum flow f^+ in G_f, and
- combine initial feasible flow f and f^+ to get the maximum flow f^* of original network $\mathcal{N}$ with lower and upper capacity bounds [exercise].

Networks with Lower Capacity Bounds

Generalized Maximum Flow / Minimum Cut Theorem:

Definition 31 (Capacity of a Cut [Extension])

The capacity of a $s-t$ cut $(S,\overline{S})$, $s \in S$, $t \in \overline{S}$, in a flow network $\mathcal{N}$ with nonnegative lower bounds is defined as follows:

$$\varsigma(S,\overline{S}) = \varsigma^U(S,\overline{S}) - \varsigma^L(\overline{S},S)$$

Networks with Lower Capacity Bounds

Generalized Maximum Flow / Minimum Cut Theorem:

Remember:

$$|f| = f(S,\overline{S}) - f(\overline{S},S)$$

Substitute flow by the corresponding capacity bounds:

$$f(u,v) \le \varsigma^U(u,v) \qquad \varsigma^L(v,u) \le f(v,u)$$

$$|f| \le \varsigma^U(S,\overline{S}) - \varsigma^L(\overline{S},S) = \varsigma(S,\overline{S})$$

Networks with Lower Capacity Bounds

Generalized Maximum Flow / Minimum Cut Theorem:

Optimality criterion for maximum flow: No augmenting $s-t$ path in G_f
$\Rightarrow$ there exists a $s-t$ cut $(S,\overline{S})$ with all $r_f(u,v)=0$, $u \in S$, $v \in \overline{S}$:

$$r_f(u,v) = \left(\varsigma^U(u,v) - f(u,v)\right) + \left(f(v,u) - \varsigma^L(v,u)\right)$$

$$r_f(u,v) = 0 \quad \Rightarrow \quad f(u,v) = \varsigma^U(u,v) \ \wedge \ f(v,u) = \varsigma^L(v,u) \quad \Rightarrow$$

Theorem 32 (Generalized Max-Flow / Min-Cut)

Let f be a flow in $\mathcal{N}$, $\varsigma(S,\overline{S})$ defined as above: The maximum value of flow from s to t equals the minimum capacity among all $s-t$ cuts:

$$|f^*| = \min_S \varsigma(S,\overline{S}) = \min_S(\varsigma^U(S,\overline{S}) - \varsigma^L(\overline{S},S)).$$

Note: This implies that $\varsigma(S,\overline{S}) \ge 0$ for all cuts S.

Networks with Lower Capacity Bounds

Phase 1 (Feasible Flow):

Transformation of the maximum flow problem into a circulation problem:
New arc (t,s) with capacities $[0,+\infty]$.

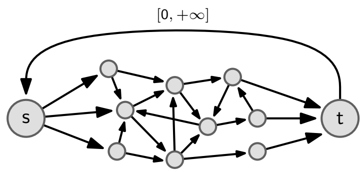


Figure: Circulation problem.

Note:

- Feasible flow $\rightarrow$ feasible circulation, but
- feasible circulation $\overset{?}{\rightarrow}$ feasible flow?

Networks with Lower Capacity Bounds

Phase 1 (Feasible Flow):

Circulation Problem

In a feasible circulation a flow f satisfies the following constraints:

$$f(u,V) - f(V,u) = 0 \qquad \forall u \in V$$

$$\varsigma^L(u,v) \le f(u,v) \le \varsigma^U(u,v) \qquad \forall (u,v) \in A$$

Note: The flow conservation contraints now hold for every node $v \in V$, including s and t.

Networks with Lower Capacity Bounds

Phase 1 (Feasible Flow): Alternative 1

Replace $f(u,v)$ by $f'(u,v)+\varsigma^L(u,v)$ in the flow conservation constraints:

$$\left(f'(u,V)+\varsigma^L(u,V)\right) - \left(f'(V,u)+\varsigma^L(V,u)\right) = 0$$

$$f'(u,V) - f'(V,u) = b(u)$$

with

$$b(u) = \varsigma^L(V,u) - \varsigma^L(u,V) \qquad \forall u \in V$$

Networks with Lower Capacity Bounds

Phase 1 (Feasible Flow): Alternative 1

This way the lower capacity bounds are removed,

$$0 \le f'(u,v) \le \varsigma^U(u,v) - \varsigma^L(u,v) \qquad \forall (u,v) \in A$$

and supplies / demands $b(\cdot)$ are introduced.

Note: $\sum_{u \in V} b(u) = 0$, since each $\varsigma^L(u,v)$ appears twice – once positive and once negative – in this expression.

$\rightarrow$ There are algorithms to handle multiple sources / sinks.

Networks with Lower Capacity Bounds

Phase 1 (Feasible Flow): Alternative 2

Theorem 33 (Circulation Feasibility Conditions)

A circulation problem with nonnegative lower bounds on arc flows is feasible iff for every arbitrary set $S \subset V$, $S \neq \emptyset$, $\overline{S} = V \setminus S$, the following condition holds:

$$\varsigma^L(\overline{S},S) \le \varsigma^U(S,\overline{S})$$

Note: Relation to generalized max-flow / min-cut theorem!
$(0 \le \varsigma^U(S,\overline{S}) - \varsigma^L(\overline{S},S) = \varsigma(S,\overline{S}))$

Networks with Lower Capacity Bounds

Phase 1 (Feasible Flow): Alternative 2

Theorem 33 is a necessary condition:

$$f(S,\overline{S}) - f(\overline{S},S) = 0$$

(generalization of the flow conservation conditions).

Using $f(u,v) \leq \varsigma^U(u,v)$ and $f(v,u) \geq \varsigma^L(v,u)$:

$$\varsigma^L(\overline{S},S) \leq \varsigma^U(S,\overline{S}).$$

Networks with Lower Capacity Bounds

Phase 1 (Feasible Flow): Alternative 2

Algorithmic proof: Theorem 33 is a sufficient condition:

Definition 34 (Feasible / Infeasible Arc)

In respect to a flow f an arc (u,v) is called infeasible if $f(u,v) < \varsigma^L(u,v)$, otherwise it is a feasible arc, i.e., $\varsigma^L(u,v) \leq f(u,v)$.

Basic idea:
Start with a flow fulfilling flow conservation conditions, but violating lower capacity bounds $\rightarrow$ transform flow (while still ensuring flow conservation and upper capacity bounds) – if possible – into circulation satisfying also the lower capacity bounds.

Networks with Lower Capacity Bounds

Phase 1 (Feasible Flow): Alternative 2

Algorithmic proof: Theorem 33 is a sufficient condition:

Definition 34 (Feasible / Infeasible Arc)

In respect to a flow f an arc (u,v) is called infeasible if $f(u,v) < \varsigma^L(u,v)$, otherwise it is a feasible arc, i.e., $\varsigma^L(u,v) \leq f(u,v)$.

Computation of residual capacities:

- If arc (v,u) is feasible:
  $r_f(u,v) = \left(\varsigma^U(u,v) - f(u,v)\right) + \left(f(v,u) - \varsigma^L(v,u)\right)$.
- If arc (v,u) is infeasible:
  $r_f(u,v) = \varsigma^U(u,v) - f(u,v)$.

Networks with Lower Capacity Bounds

Phase 1 (Feasible Flow): Alternative 2

Algorithmic proof: Theorem 33 is a sufficient condition:

Function `feasible_circulation`

$f(u,v) \leftarrow 0 \quad \forall (u,v) \in A$; `/* initialization */`

while $\exists$ *an infeasible arc* $(u,v) \in G_f$ **do** `/* algorithm */`
    find directed path $P(v,u)$ in G_f;
    if $\nexists P(v,u)$ **then return** $S = \{v \cup$ nodes reachable from v in $G_f\}$;
    $P(v,u) \cup (u,v) \rightarrow$ augmenting cycle in G_f;
    augment flow along $P(v,u) \cup (u,v) \rightarrow$
        (u,v) becomes feasible, or cycle cannot carry more flow;

return f;

Networks with Lower Capacity Bounds

Phase 1 (Feasible Flow): Alternative 2
Algorithmic proof: Theorem 33 is a sufficient condition:

- Algorithm terminates returning feasible circulation. ✓
- Algorithm returns set S, infeasible arc (u, v):
  - Let $\overline{S} = V \setminus S$, $x \in S$, $y \in \overline{S}$. $r_f(x, y) = 0$ in G_f, otherwise y could be reached from $v \Rightarrow f(x, y) = \varsigma^U(x, y)$ and $f(y, x) \leq \varsigma^L(y, x) \Rightarrow$ it is not possible to send more flow out of S.
  - $u \in \overline{S}$ (no path from v to u), $v \in S$, (u, v) infeasible $\Rightarrow$ $f(u, v) < \varsigma^L(u, v)$, i.e., at least one arc $(\overline{S}, S)$ requires to send more flow into $S \Rightarrow$

$$\begin{aligned} f(\overline{S}, S) &= f(S, \overline{S}) \quad \text{(flow conservation)} \\ \Rightarrow \varsigma^L(\overline{S}, S) &> \varsigma^U(S, \overline{S}) \end{aligned}$$

$\Rightarrow$ ↯ contradiction to conditions of Theorem 33. □

Networks with Lower Capacity Bounds

Problem: Feasible circulation $\stackrel{?}{\rightarrow}$ feasible $s - t$ flow?

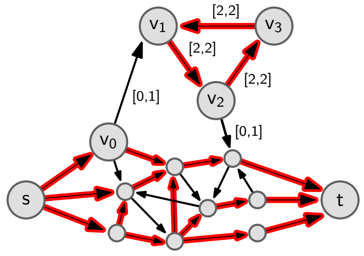


Figure: Network $\mathcal{N}$ with a feasible circulation, but no feasible $s - t$ flow: It is not possible to bring the required flow from s to the circle $v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_1$.

- `feasible_circulation()`:
  It has to be ensured that arc (t, s) is part of the directed path from v to u, i.e., $P(v, u) = v \rightsquigarrow t \rightarrow s \rightsquigarrow u$.

MINIMUM COST FLOW IN NETWORKS

Minimum Cost Flow: Introduction

Definition 35 (Minimum Cost Flow)

Given a directed graph $G(V, A)$ with costs $c(u, v)$ and a capacity $\varsigma(u, v)$ associated with each arc $(u, v) \in A$, the minimum cost flow problem can be stated as follows:

$$\text{Minimize} \quad z(f) = \sum_{(u,v)\in A} c(u, v) \cdot f(u, v)$$

subject to:

$$f(u, V) - f(V, u) = b(u) \qquad \forall u \in V$$

$$0 \leq f(u, v) \leq \varsigma(u, v) \qquad \forall (u, v) \in A.$$

Minimum Cost Flow: Introduction

Definition 36 (Supply / Demand)

A value $b(v) > 0$ denotes a supply of $b(v)$ units of flow at node v, whereas a value $b(v) < 0$ denotes a demand at this node.

Assumption

All data, i.e., costs, capacity, supply, and demand, are integral.

Assumption

All arc costs are nonnegative, or at least there is no directed negative cost cycle of infinite capacity.

Minimum Cost Flow: Introduction

Minimum cost flows arise in a lot of different applications, respectively various (sub-)problems can be reformulated as a minimum cost flow problem:

- Shipping and distribution: the transportation problem (e.g. plants with supplies $\rightarrow$ warehouses with demands, minimizing the shipping costs).
- Optimal loading of a hopping airplane.
- Reconstruction of organs (e.g. ventricle) based on x-ray projections.
- Scheduling with deferral costs (uniform processing times of jobs).
- Efficient solving of linear programs with special structure ($0-1$ matrix, consecutive 1's in columns).
- ...

Minimum Cost Flow: Introduction

Necessary Condition for Feasibility

The supplies and demands have to satisfy $\sum_{v \in V} b(v) = 0$.

(Is it also sufficient? No!)

Test for feasibility:

- Introduce two new, additional nodes s and t.
- Introduce new arcs:
  - For every node v with $b(v) > 0$: $A = A \cup (s, v)$, $\varsigma(s, v) = b(v)$.
  - For every node v with $b(v) < 0$: $A = A \cup (v, t)$, $\varsigma(v, t) = -b(v)$.
- Solve a maximum flow problem on the modified graph. If all the arcs $(s, \cdot)$ and $(\cdot, t)$ are saturated, there exists a feasible solution to the original minimum cost flow problem.

Minimum Cost Flow: Introduction

Definition 37 (Residual Network)

Given a flow f, each arc $(u, v) \in A$ is replaced in the residual network G_f by two arcs:

- An arc (u, v) with costs $c(u, v)$ and a residual capacity $r_f(u, v) = \varsigma(u, v) - f(u, v)$, and
- an arc (v, u) with costs $c(v, u) = -c(u, v)$ and $r_f(v, u) = f(u, v)$.

Note: r_f is always ≥ 0.

Minimum Cost Flow: Optimality Conditions

Negative Cycle Optimality Condition:

A feasible solution f^* is an optimal solution of the minimum cost flow problem iff the residual network G_{f^*} contains no directed negative cost cycle.

Sketch of Proof:

- Sending flow along a cycle does not change the flow conservation conditions at any node of the network.
- The residual network G_f only contains arcs that can carry additional flow, i.e., it is possible to send flow along such arcs without violating the capacity bounds.
- Consequence: When sending flow along a negative cost cycle in G_f the flow f remains feasible but the costs can be reduced.

Minimum Cost Flow: Optimality Conditions

Reduced Costs Optimality Condition:

Observation

Optimality condition for shortest path regarding costs:

$$c^d(u,v) = d(u) + c(u,v) - d(v) \geq 0 \qquad \forall (u,v) \in A$$

$c^d(u,v)$ is referred to as the reduced costs for arc (u,v).

Interpretation:

$c^d(u,v)$ measures the costs of the arc (u,v) relative to the shortest path distances $d(u)$ and $d(v)$.

Note: If (u,v) is part of a shortest path from a node s to v, then $c^d(u,v) = 0$, otherwise $c^d(u,v) > 0$.

Minimum Cost Flow: Optimality Conditions

Reduced Costs Optimality Condition:

Definition 38 (Node Potential)

We associate a potential $\pi(v) \in \mathbb{R}$ to each node $v \in V$.

Interpretation: $\pi(v)$ is the linear programming dual variable corresponding to the flow conservation condition at node v.

Definition 39 (Reduced Costs (Minimum Cost Flow))

Based on node potentials $\pi(\cdot)$, the reduced cost of an arc (u,v) in G or G_f is defined as follows:

$$c^\pi(u,v) = c(u,v) - \pi(u) + \pi(v).$$

Minimum Cost Flow: Optimality Conditions

Reduced Costs Optimality Condition:

Lemma 40 (Path and Node Potentials)

For any directed path P from u to v the following equation holds:

$$\sum_{(i,j)\in P} c^\pi(i,j) = \sum_{(i,j)\in P} c(i,j) - \pi(u) + \pi(v).$$

Lemma 41 (Cycle and Node Potentials)

For any directed cycle W the following equation holds:

$$\sum_{(i,j)\in W} c^\pi(i,j) = \sum_{(i,j)\in W} c(i,j).$$

Consequence: $\exists$ negative cost cycle with respect to $c(\cdot) \Leftrightarrow \exists$ negative cost cycle with respect to $c^\pi(\cdot)$.

Minimum Cost Flow: Optimality Conditions

Reduced Costs Optimality Condition:

A feasible solution f^* is an optimal solution of the minimum cost flow problem iff some set of node potentials $\pi(\cdot)$ satisfy the reduced cost optimality conditions:

$$c^\pi(u,v) \geq 0 \qquad \forall (u,v) \in G_{f^*}$$

Proof:

$\Leftarrow$: Direct consequence of the negative cycle optimality condition and the preceding lemma.

$\Rightarrow$: Now assume a solution f^* contains no negative cycle in G_{f^*} $\Rightarrow$ let $d(\cdot)$ be the shortest path distance from a fixed node to all other nodes in G_{f^*} $\Rightarrow$ $d(v) \leq d(u) + c(u,v)$ $\Rightarrow$ $c(u,v) - (-d(u)) + (-d(v)) \geq 0$ $\Rightarrow$ with $\pi(\cdot) = -d(\cdot)$: $c(u,v) - \pi(u) + \pi(v) = c^\pi(u,v) \geq 0$. $\square$

Minimum Cost Flow: Optimality Conditions

Reduced Costs Optimality Condition:

Economic interpretation:

- $c(u,v)$: cost to send one unit of flow from u to v,
- $\mu(u)$: cost to obtain one unit of flow at u $\Rightarrow$
- $\mu(u) + c(u,v)$: cost of one unit of flow at v in case arc (u,v) is used to transport it.
- $\mu(v) \leq \mu(u) + c(u,v)$, $\mu(u) = -\pi(u)$ $\Leftrightarrow$ $c(u,v) - \pi(u) + \pi(v) \geq 0$:
  $\mu(v) = \mu(u) + c(u,v)$: flow to v uses arc (u,v).
  $\mu(v) < \mu(u) + c(u,v)$: there is a cheaper way to get the flow to v.

Minimum Cost Flow: Optimality Conditions

Complementary Slackness Optimality Condition:

A feasible solution f^* is an optimal solution of the minimum cost flow problem iff for some set of node potentials $\pi(\cdot)$ the reduced costs and flow values satisfy the following complementary slackness optimality conditions for every $(u,v) \in A$ (original network):

- If $c^\pi(u,v) > 0$, then $f^*(u,v) = 0$.
- If $0 < f^*(u,v) < \varsigma(u,v)$, then $c^\pi(u,v) = 0$.
- If $c^\pi(u,v) < 0$, then $f^*(u,v) = \varsigma(u,v)$.

Minimum Cost Flow: Optimality Conditions

Complementary Slackness Optimality Condition:

Sketch of Proof:

$\Rightarrow$: Node potentials $\pi(\cdot)$ and the flow f^* satisfy the reduced cost optimality conditions ($c^\pi(u,v) \geq 0 \;\; \forall (u,v) \in G_{f^*}$) $\Rightarrow$ they have to satisfy complementary slackness optimality condition:

- If $c^\pi(u,v) > 0 \Rightarrow$ arc $(v,u) \notin G_{f^*}$, because $c^\pi(u,v) = c(u,v) - \pi(u) + \pi(v) = -c^\pi(v,u) \Rightarrow c^\pi(v,u) < 0 \Rightarrow ↯$ to optimality condition $\Rightarrow f^*(u,v) = 0$.
- If $0 < f^*(u,v) < \varsigma(u,v)$, then G_{f^*} contains both arcs (u,v) and $(v,u) \Rightarrow c^\pi(u,v) \geq 0,\; c^\pi(v,u) \geq 0,\; c^\pi(u,v) = -c^\pi(v,u) \Rightarrow c^\pi(u,v) = c^\pi(v,u) = 0$.
- If $c^\pi(u,v) < 0$, arc $(u,v) \notin G_{f^*}$ (otherwise ↯ to assumption) $\Rightarrow f^*(u,v) = \varsigma(u,v)$

Minimum Cost Flow: Algorithms

Cycle-Canceling Algorithm:

Basic Idea: Establish feasible flow; keep flow feasible but improve costs until optimum reached.

Procedure `Cycle-Canceling()`

establish feasible flow f in network; `/* initialization */`

while $\exists$ *negative cost cycle* W *in* G_f **do** `/* algorithm */`

  $x \leftarrow$ minimum residual capacity along W;

  augment flow by value x along W;

  update G_f;

Minimum Cost Flow: Algorithms

Cycle-Canceling Algorithm:

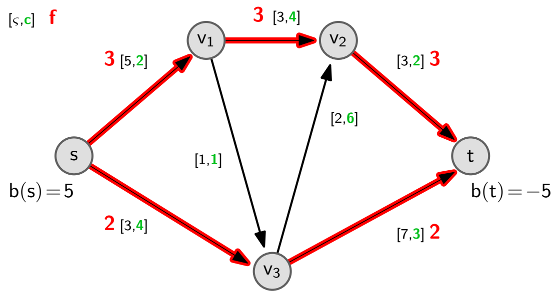


Figure: Initialization: A feasible flow from s to t in network $\mathcal{N}$.

Minimum Cost Flow: Algorithms

Cycle-Canceling Algorithm:

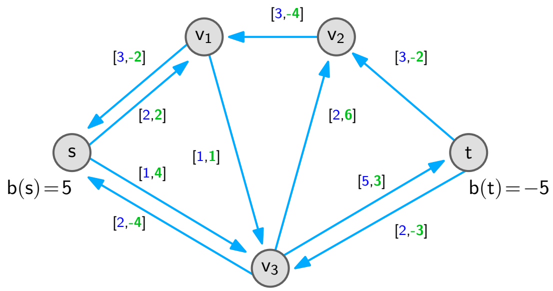


Figure: Residual network G_f.

Minimum Cost Flow: Algorithms

Cycle-Canceling Algorithm:

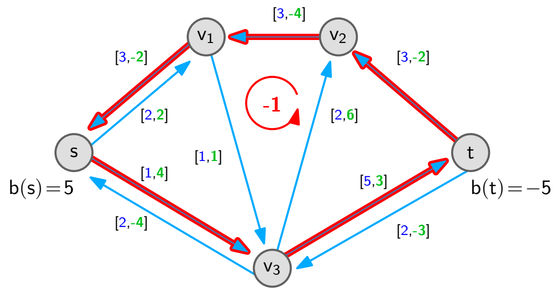


Figure: Negative cost cycle: $t \to v_2 \to v_1 \to s \to v_3 \to t$, costs: -1.

Minimum Cost Flow: Algorithms

Cycle-Canceling Algorithm:

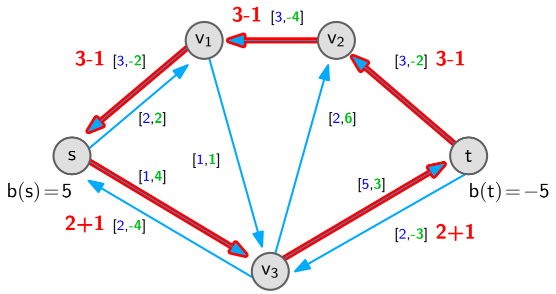


Figure: Minimum residual capacity: $r_f(s, v_3) = 1$; augment flow along cycle.

Minimum Cost Flow: Algorithms

Cycle-Canceling Algorithm:

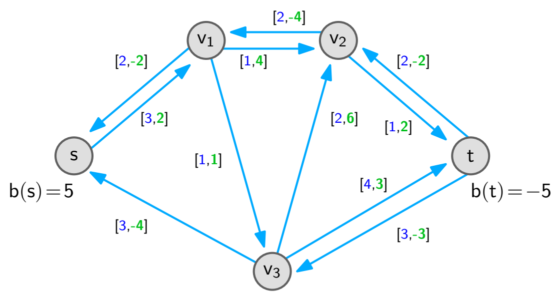


Figure: Resulting residual network G_f.

Minimum Cost Flow: Algorithms

Cycle-Canceling Algorithm:

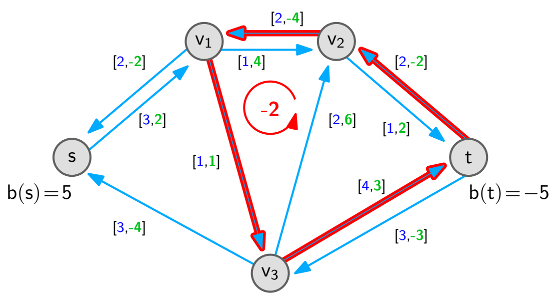


Figure: Negative cost cycle: $t \to v_2 \to v_1 \to v_3 \to t$, costs: -2.

Minimum Cost Flow: Algorithms

Cycle-Canceling Algorithm:

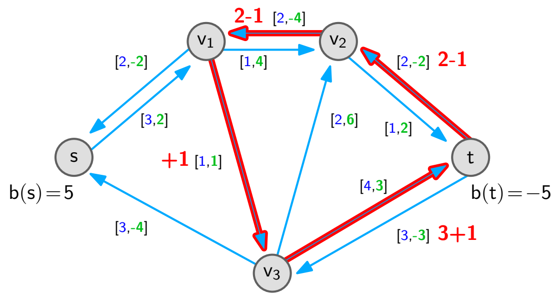


Figure: Minimum residual capacity: $r_f(v_1, v_3) = 1$; augment flow along cycle.

Minimum Cost Flow: Algorithms

Cycle-Canceling Algorithm:

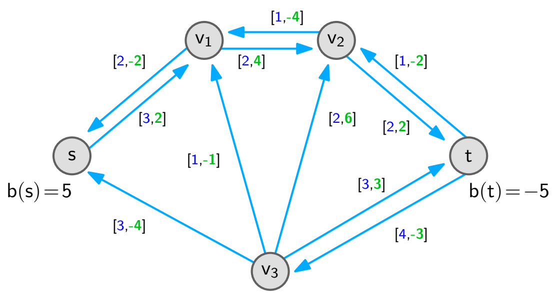


Figure: Resulting residual network G_f; no negative cost cycle.

Minimum Cost Flow: Algorithms

Cycle-Canceling Algorithm:

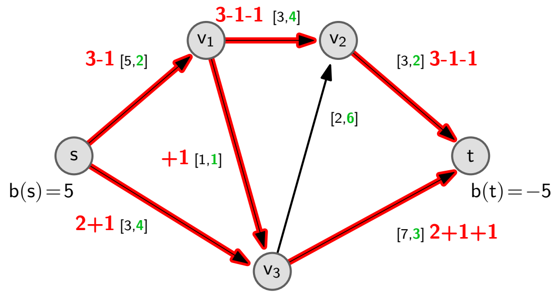


Figure: Original flow and flows augmented along negative cost cycles.

Minimum Cost Flow: Algorithms

Cycle-Canceling Algorithm:

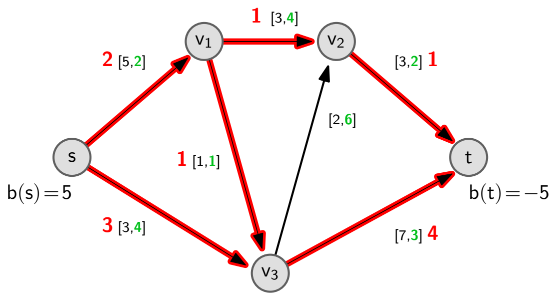


Figure: Minimum cost flow in network $\mathcal{N}$.

Minimum Cost Flow: Algorithms

Cycle-Canceling Algorithm:

Definition 42 (C, U)

$C = \max\{c(u,v) : (u,v) \in A\}$.
$U = \max\{\varsigma(u,v) : (u,v) \in A \land \varsigma(u,v) < \infty\}$.

Running time:

- Establishing a feasible flow: $O(n^2 \cdot m)$ (preflow-push algorithm).
- Number of iterations: $O(m \cdot C \cdot U)$ (integrality condition).
- Identifying a negative cost cycle: $O(n \cdot m)$ (shortest path algorithm, e.g. Bellman-Ford).

$\Rightarrow O(n \cdot m^2 \cdot C \cdot U)$

Variation: Network simplex algorithm: Widely considered one of the fastest algorithms in practice; identifies a negative cost cycle in $O(m)$ (but objective function cannot be reduced in every iteration).

Minimum Cost Flow: Algorithms

Successive Shortest Path Algorithm:

Basic Idea: Start with "solution" satisfying reduced costs optimality condition, but not flow conservation (excess / deficit $\rightarrow$ "pseudoflow"); keep optimality condition and transform pseudoflow into feasible flow.

Definition 43 (E, D)

$$e_f(u) = b(u) - f(u, V) + f(V, u) \qquad \forall u \in V$$

E = Set of nodes with excess ($e_f(\cdot) > 0$).
D = Set of nodes with deficit ($e_f(\cdot) < 0$).

Minimum Cost Flow: Algorithms

Successive Shortest Path Algorithm:

Procedure `Successive Shortest Path()`

```
f(·) = 0, π(·) = 0;                                   /* initialization */;
e(v) = b(v), ∀v ∈ V; initialize sets E and D;
while E ≠ ∅ do                                        /* algorithm */
    select a node u ∈ E and a node v ∈ D;
    compute shortest path distances d(·) from u to all other nodes in G_f
        with respect to reduced costs c^π(·);
    P ← shortest path from u to v;
    π(·) = π(·) − d(·);
    x = min{e(u), −e(v), min{r_f(i,j) : (i,j) ∈ P}};
    augment flow of value x along P;
    update f(·), G_f, E, D, c^π(·);
```


Minimum Cost Flow: Algorithms

Successive Shortest Path Algorithm:

Running time:

- Number of iterations: $O(n \cdot U)$ (U: upper bound on largest supply).
- Shortest path algorithm: $S(n, m)$ (nonnegative arc costs).

$\Rightarrow O(n \cdot U \cdot S(n, m))$