Combinatorial Mathematics

Mong-Jen Kao (高孟駿)

Monday 18:30 – 21:20

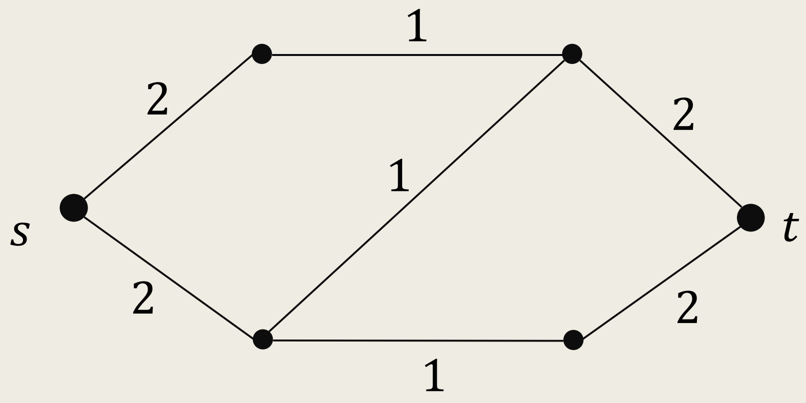
Outline

- The Problem Model
  - Weak-Duality between Max-Flow and Min-Cut
- The Ford-Fulkerson Algorithm
- Some Efficient Augmenting Path Algorithms for Max-Flow
  - Capacity Scaling, Edmonds-Karp
- Concluding Notes

The Network Flow Problem

Basic Definitions

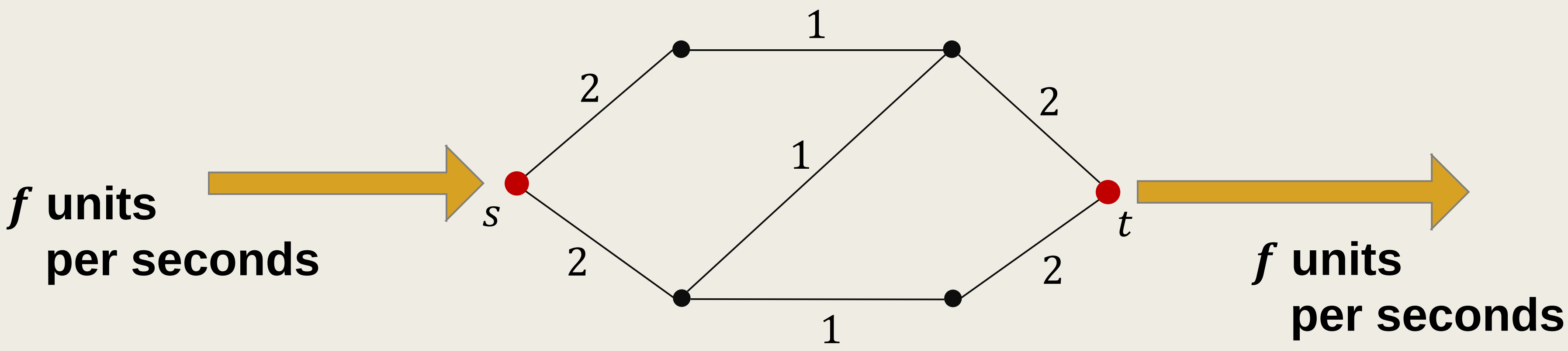
- A network is an undirected graph $G = (V, E)$ with
  - ***Edge capacity*** $\boldsymbol{c_{u,v}} \in \mathbb{R}^{\geq 0}$ for each $(u, v) \in E$,
  - A ***source vertex*** $\boldsymbol{s} \in V$, and
  - A ***sink vertex*** $\boldsymbol{t} \in V$.



The network flow problem was originally defined on *directed graphs*.
In this lecture, let's assume undirected graphs for simplicity.

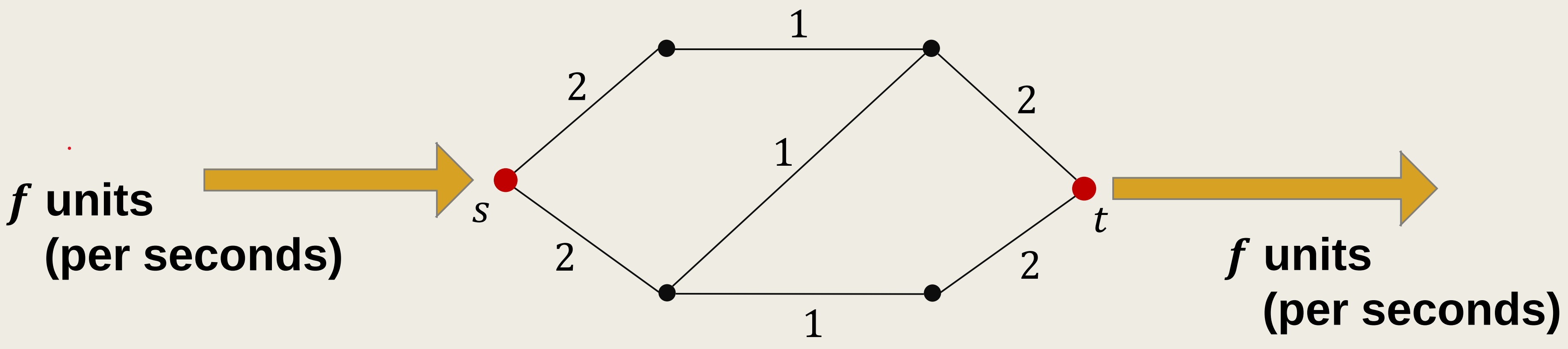
The Problem Model

Flow can be ***water***, ***network packages***, ***gasoline***, etc.



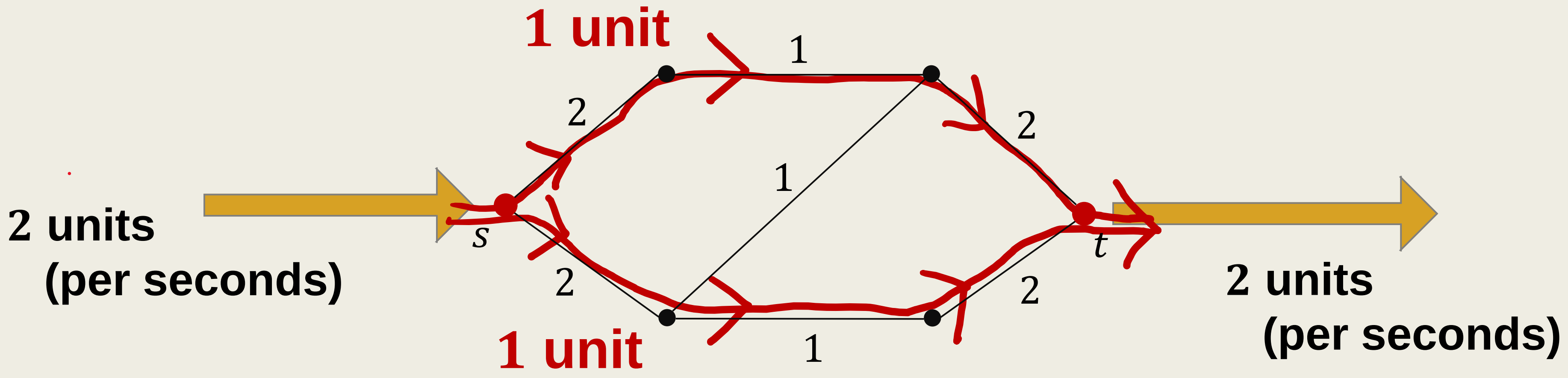
- ***Flow*** is ***sent into*** the network via ***the source vertex*** $\boldsymbol{s}$, flowed through the pipes of the network, and then ***exited*** from the network via ***the sink vertex*** $\boldsymbol{t}$.

The Problem Model



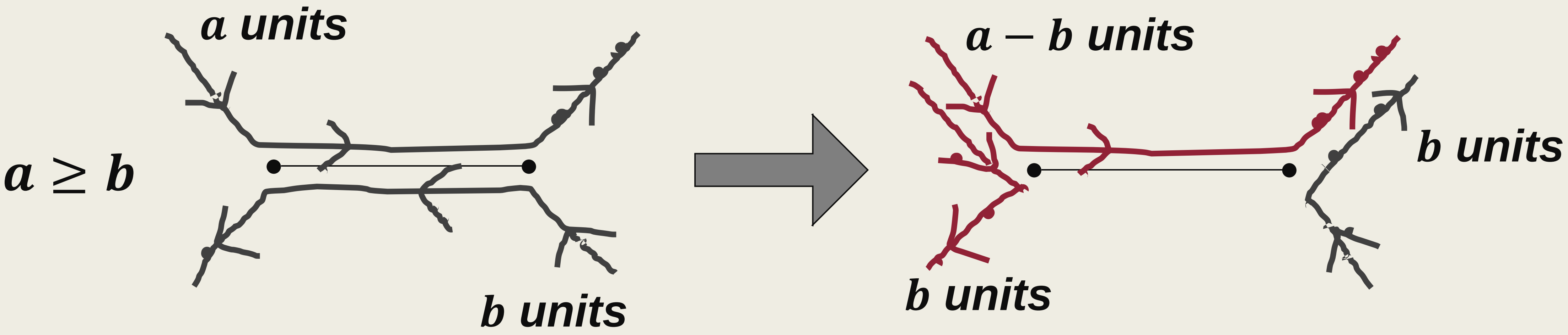
- The ***edges*** in the network are pipes ***with limited capacity***, and allow flow to be sent ***in either directions***.

The Problem Model



■ We can decide how the flow goes in the network.

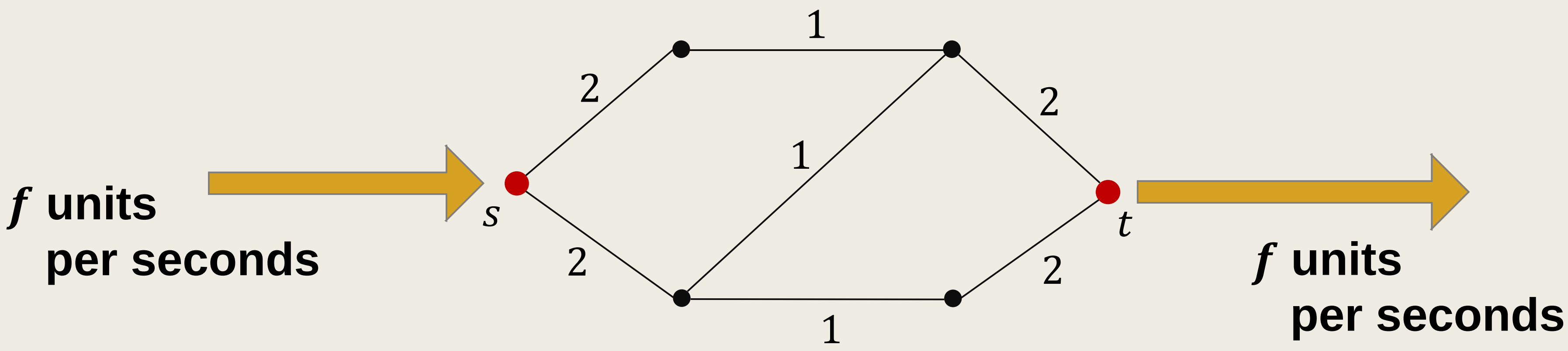
The Problem Model



- Flow sent from different directions of an edge cancels out.

Flow can be ***water***, ***network packages***, ***gasoline***, etc.

The Problem Model



- ***Question:***

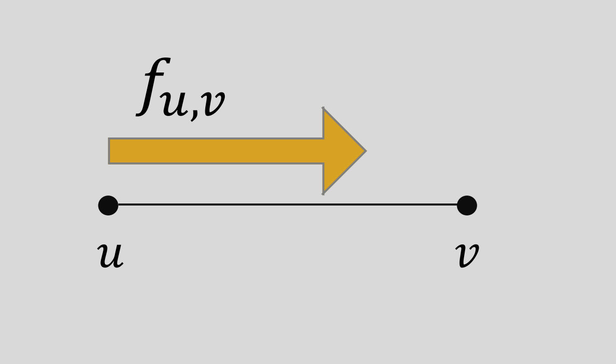
What is the ***maximum amount of flow*** that can be sent?

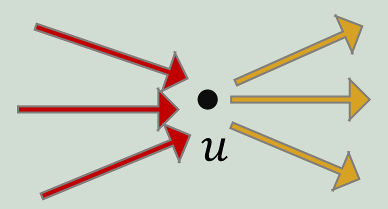
Formal Definition

- An s-t flow f is a function $f : V \times V \longrightarrow \mathbb{R}$ such that
  - $f_{u,v} = f_{v,u} = 0$, for all $(u, v) \notin E$.
  - (Symmetric) $f_{u,v} = -f_{v,u}$, for all $(u, v) \in E$.
  - (Conservation) for any $u \in V \backslash \{s, t\}$,

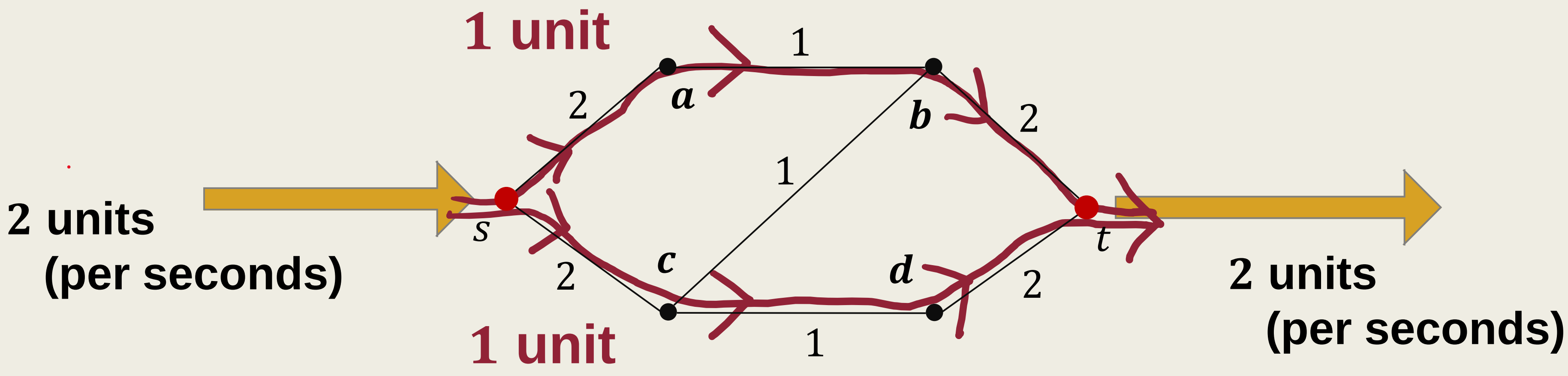
$$\sum_{\substack{v:(u,v)\in E,\\ f_{v,u}>0}} f_{v,u} = \sum_{\substack{v:(u,v)\in E,\\ f_{u,v}>0}} f_{u,v} .$$

  - $f_{s,u} \geq 0$ and $f_{u,t} \geq 0$ for all $u \in V$.





Conservation of Flow



- In this example,

$$f_{s,a} = f_{a,b} = f_{b,t} = 1, \qquad f_{s,c} = f_{c,d} = f_{d,t} = 1,$$

$$f_{a,s} = f_{b,a} = f_{t,b} = -1, \qquad f_{c,s} = f_{d,c} = f_{t,d} = -1.$$

Formal Definition

- The value of a flow function f is defined as

$$\mathrm{val}(f) \coloneqq \sum_{v:(s,v)\in E} f_{s,v} \ .$$

  - By the conservation constraint, $\mathrm{val}(f)$ is also equal to

$$\sum_{v:(v,t)\in E} f_{v,t} \ .$$

The Maximum s-t Flow Problem

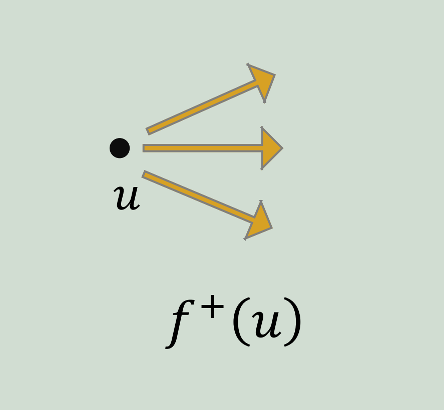
- Input :
  - A graph / flow network $G = (V, E)$ with edge capacity $c_{u,v} \in \mathbb{R}^{\geq 0}$ for all $(u, v) \in E$ and a source-sink pair $s, t \in V$.
- Output :
  - A flow function $f : E \longrightarrow \mathbb{R}^{\geq 0}$ that has the maximum value among all possible s-t flows for G.
    - That is, $\mathrm{val}(f) \geq \mathrm{val}(f')$ holds for all s-t flow f' for G.

Notations

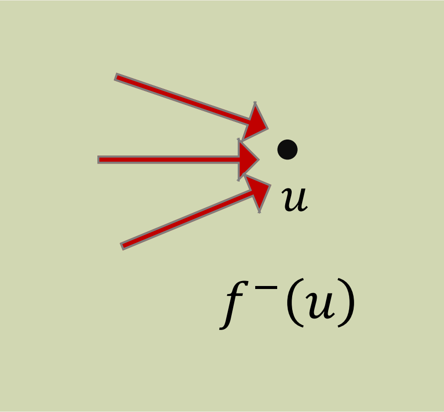
- For any vertex $u \in V$, we use

$$f^+(u) \coloneqq \sum_{\substack{v:(u,v)\in E \\ f_{u,v}>0}} f_{u,v}$$

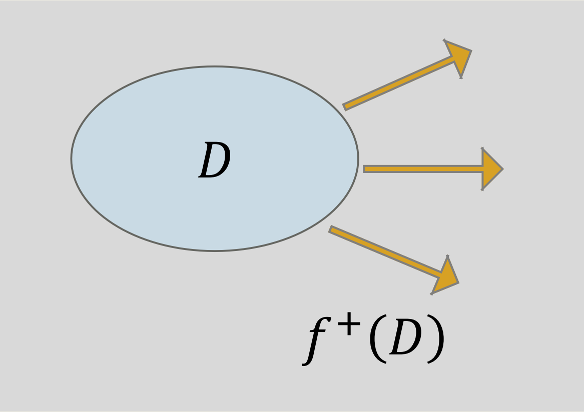
to denote the total amount of flow leaving the vertex u.



  – Similarly, we use $f^-(u) \coloneqq \sum_{\substack{v:(u,v)\in E \\ f_{v,u}>0}} f_{v,u}$ to denote the total amount of flow entering the vertex u.



Notations

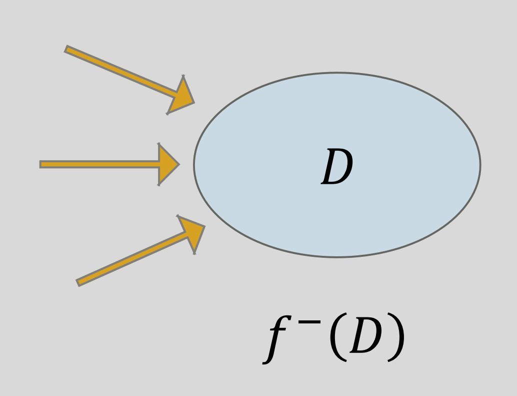


- For any $D \subseteq V$, we use

$$f^+(D) := \sum_{\substack{u \in D,\, v \in V \setminus \mathrm{D} \\ f_{u,v} > 0}} f_{u,v}$$

to denote the total amount of flow leaving the vertex set D.

  – Similarly, we use $f^-(D) := \sum_{\substack{u \in D,\, v \in V \setminus \mathrm{D} \\ f_{v,u} > 0}} f_{v,u}$ to denote the total amount of flow entering the vertex u.



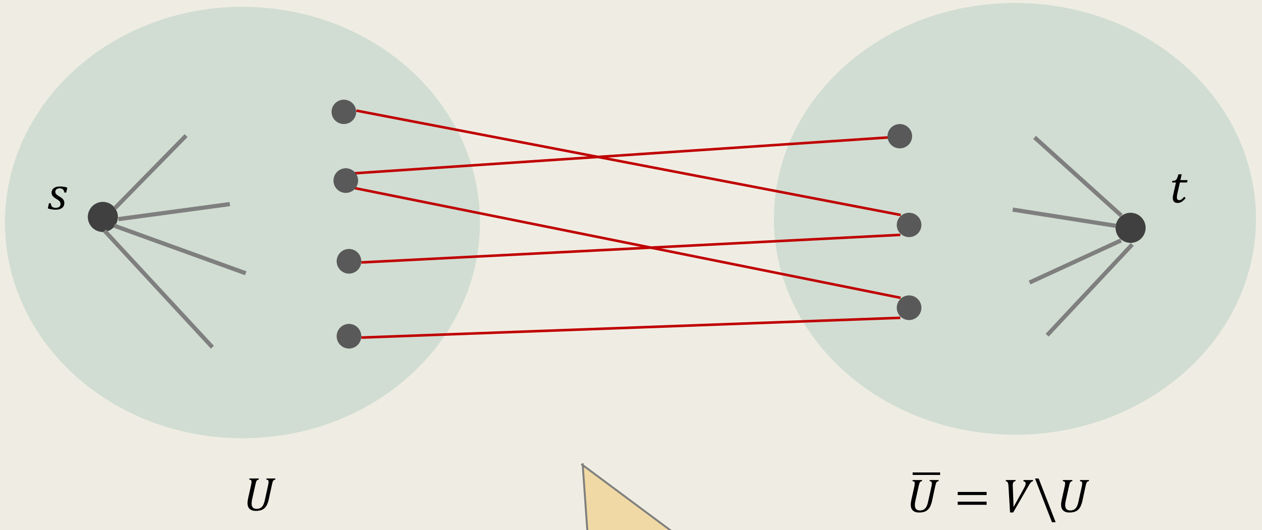
The Minimum s-t Cut Problem

The Cut for a Flow Network

- Let $G = (V, E)$ be a flow network with edge capacity (weight) $c_{u,v} \in \mathbb{R}^{\geq 0}$ for all $(u, v) \in E$ and a source-sink pair $s, t \in V$.

- **Definition**. (s-t Cut)
  - An s-t cut $C = [U, \overline{U}]$ is a partition of V into two sets $U, \overline{U}$ such that $s \in U$ and $t \in \overline{U}$.
  - Conventionally, the s-t cut $[U, \overline{U}]$ can also be referred to as the edges between U and $\overline{U}$, depending on the context.

An s-t cut $[U, \overline{U}]$



Intuitively,

an s-t cut is a set of edges,

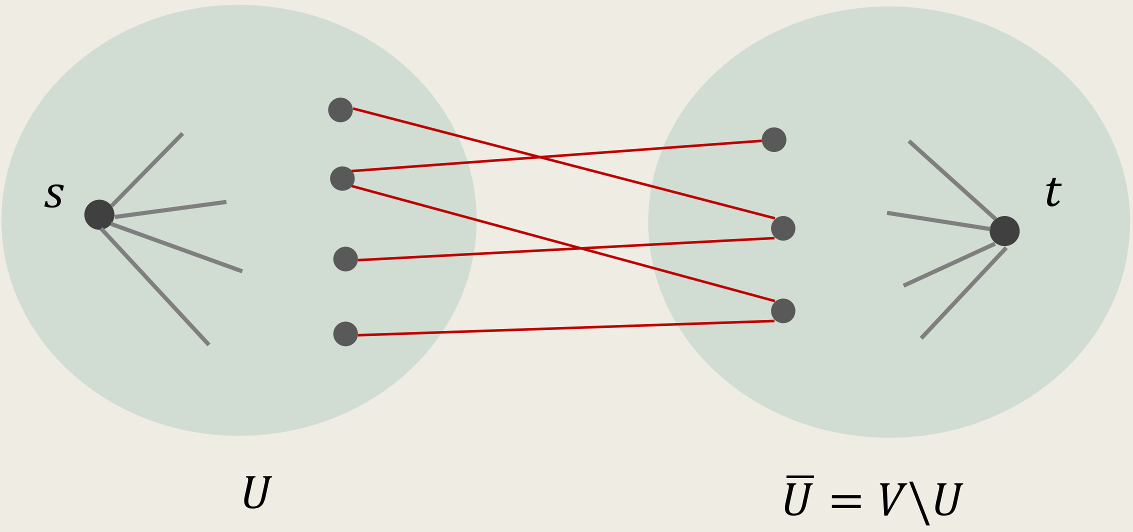
whose removal disconnects s from t.

– The ***weight of a cut*** $C = [U, \overline{U}]$ is defined to be the total weight (capacity) of the edges between U and $\overline{U}$.

- That is,

$$w(C) = \sum_{e \in C} c_e \ .$$

An s-t cut $[U, \overline{U}]$



The Minimum s-t Cut Problem

- Input :
  - A graph $G = (V, E)$ with capacity (weight) $c_{u,v} \in \mathbb{R}^{\geq 0}$ for all $(u, v) \in E$ and a source-sink pair $s, t \in V$.
- Output :
  - An s-t cut C for G that has the minimum weight among all possible s-t cut for G.
    - That is, $w(C) \leq w(C')$ holds all s-t cut C' for G.

The Weak Duality between

Maximum Flow & Minimum Cut

The maximum s-t flow is always bounded by the minimum s-t cut.

Lemma 1. (Weak Duality between Flows and Cuts)

Let $G = (V, E)$ be a graph with edge capacity $c_e \in \mathbb{R}^{\geq 0}$ for all $e \in E$,

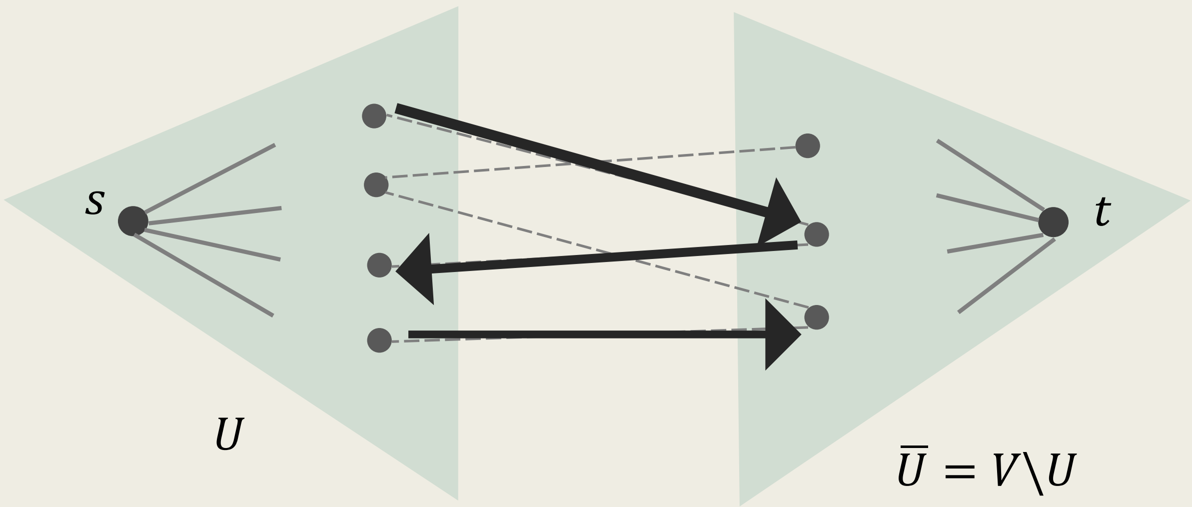
a source-sink pair $s, t \in V$,

f be an s-t flow and $C = [U, \overline{U}]$ be an s-t cut for G.

Then, $\mathrm{val}(f) \leq w(C)$, i.e., $$\sum_{v \in V : (s,v) \in E} f_{s,v} \leq \sum_{e \in C} c_e .$$

- The proof for Lemma 1 is straightforward.
  - We have

$$\mathrm{val}(f) \;=\; f^+(U) - f^-(U) \;\leq\; f^+(U) \;\leq\; \sum_{e \in C} c_e \,.$$



Remarks.

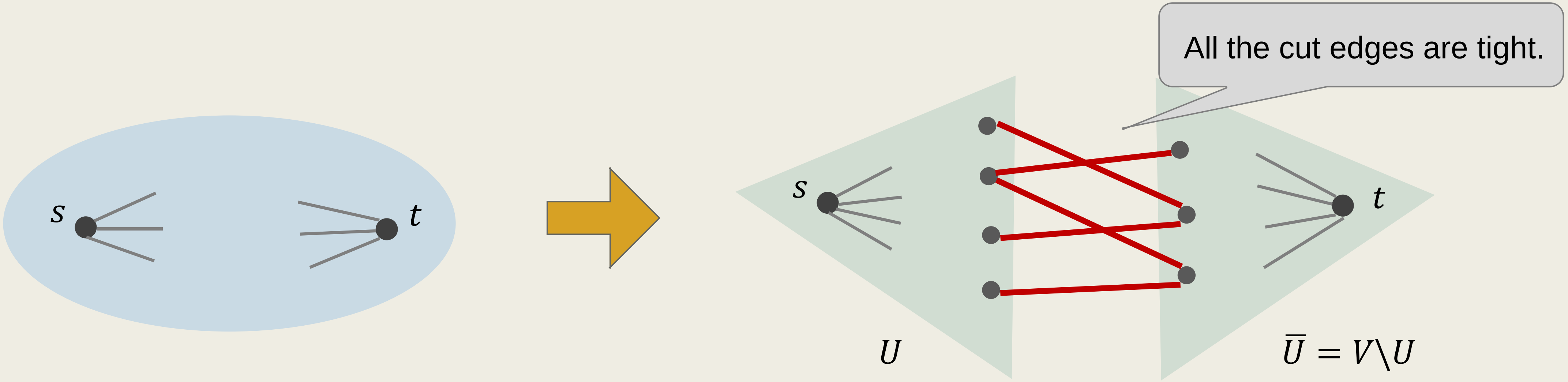
- Lemma 1 implies that,
  - If $\mathrm{val}(f) = w(C)$ holds for some f and C, then they are both optimal.
  - In this case, we say that f and C *witnesses* the optimality of each other.

The Residual Network G_f and

The Ford-Fulkerson Algorithm

Computing the Maximum Flow

- A simple greedy algorithm
  - Start with a trivial flow $f = 0$.
  - Iteratively increase the current flow to make the value larger, until no more flow can be sent.
  - Then, we have a cut with an equal weight for the network.

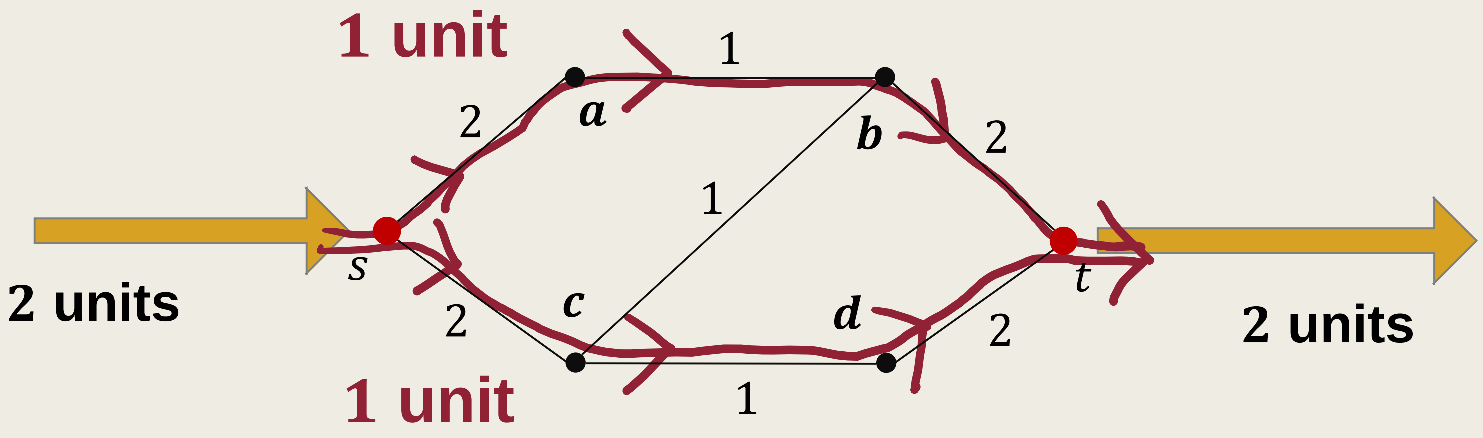


The Residual Graph G_f

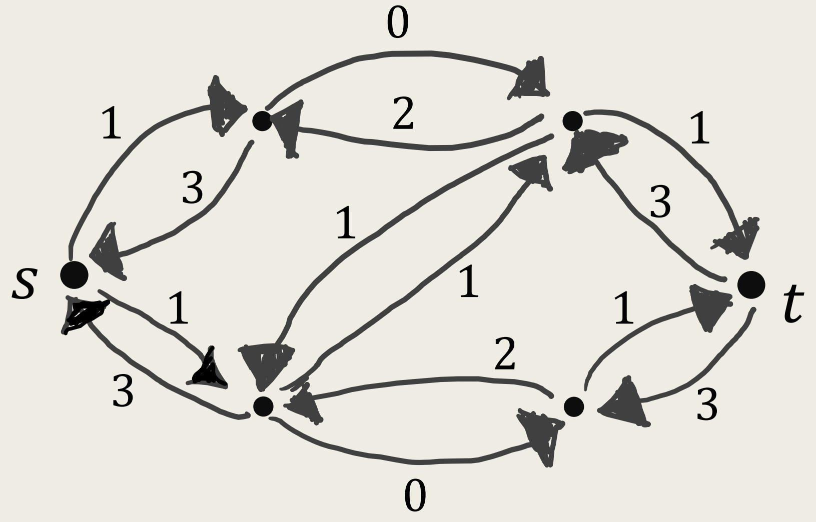
For recording the status of the flow network

- Let f be a flow function for the input graph G.
- Define the residual graph $G_f = (V, E_f)$ to be the directed graph with
  - Vertex set V,
  - (Directed) Edge set $E_f \coloneqq \{ \ (u,v) \ : \ (u,v) \in E \ \}$,
  - Capacity $c_f(u,v) \coloneqq c_{u,v} - f_{u,v}$, for each $(u,v) \in E_f$.

Intuitively, $c_f(u,v)$ is the remaining capacity on the directed edge (u,v).



1 unit
1
2
a
b
2
1
2 units
s
c
d
t
2 units
2
2
1 unit
1



0
1
2
1
3
3
1
s
1
1
t
1
2
3
3
0

Augmenting Paths in the Residual Graph G_f

- Let G_f be a residual graph with edge capacity c_f.
  - An s-t path

$$P = v_0\, v_1\, v_2\, \cdots\, v_k$$

    with $s = v_0$ and $t = v_k$ is said to be an f-augmenting path if

    - $c_f(v_i, v_{i+1}) > 0$, for all $0 \le i < k$.

The ***residual capacity*** along the path is $> \mathbf{0}$.

- Let G_f be a residual graph with edge capacity c_f.
  - An s-t path

$$P = v_0\, v_1\, v_2\, \cdots\, v_k$$

    with $s = v_0$ and $t = v_k$ is said to be an f-augmenting path if

    - $c_f(v_i, v_{i+1}) > 0$, for all $0 \le i < k$.

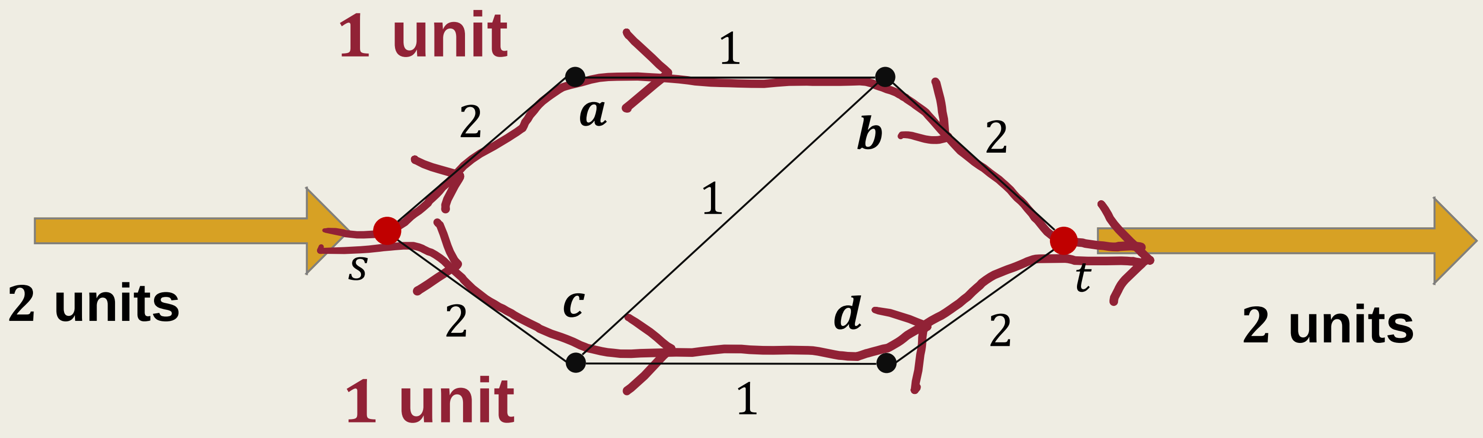
The ***residual capacity*** along the path is $> \mathbf{0}$.

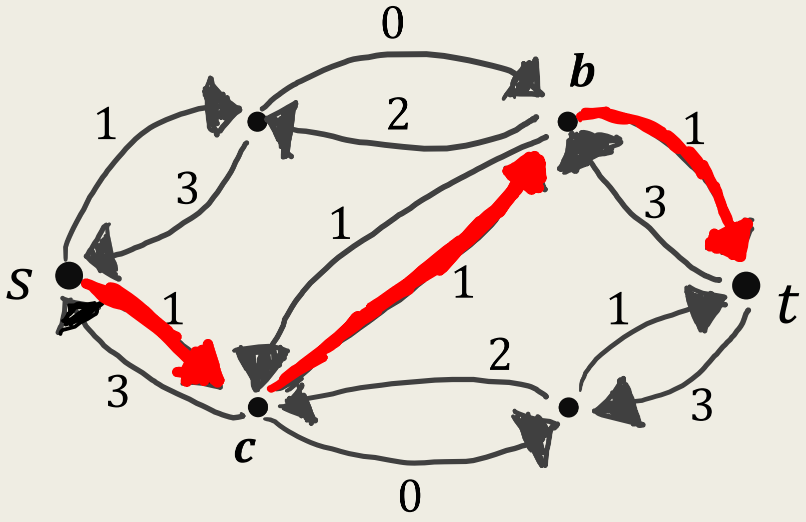
  - Define

$$\Delta_P \coloneqq \min_{0 \le i < k} c_f(v_i, v_{i+1})$$

    to be the minimum capacity along the path P in G_f.

An extra flow with value Δ_P can be sent along P.





$P = scbt$ with $\Delta_P = 1$

The value of f can be increased by $\Delta_P = 1$, by sending one unit of flow along $P = scbt$.

The Ford-Fulkerson Algorithm for Max-Flow

- Start with a trivial flow $f = 0$.
- Repeat the following until there exists no f-augmenting path in G_f.
  - Compute an f-augmenting path P in G_f.
  - Use P to increase f by Δ_P.
- Output f.

A Slightly More-Detailed Pseudo-Code

- $f \longleftarrow 0$.

  $\text{resCap}(u, v) = \text{resCap}(v, u) = c_{u,v}$ for all $(u, v) \in E$.

- While there exists an f-augmenting path P in G_f, do
  - For each edge $(a, b) \in P$,
    - Decrease $\text{resCap}(a, b)$ by Δ_P,
    - Increase $\text{resCap}(b, a)$ by Δ_P.

- Output f.

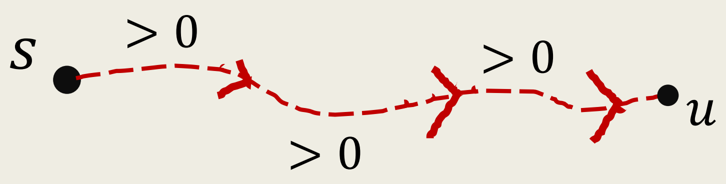
The Correctness of the Algorithm

- To prove that the Ford-Fulkerson algorithm computes a maximum s-t flow for the input graph G,
  - We show that,
    when there exists no f-augmenting path in G_f,
    G_f has an s-t cut C with weight $\mathrm{val}(f)$.
  - By Lemma 1, both C and f are optimal.

The Correctness of the Algorithm

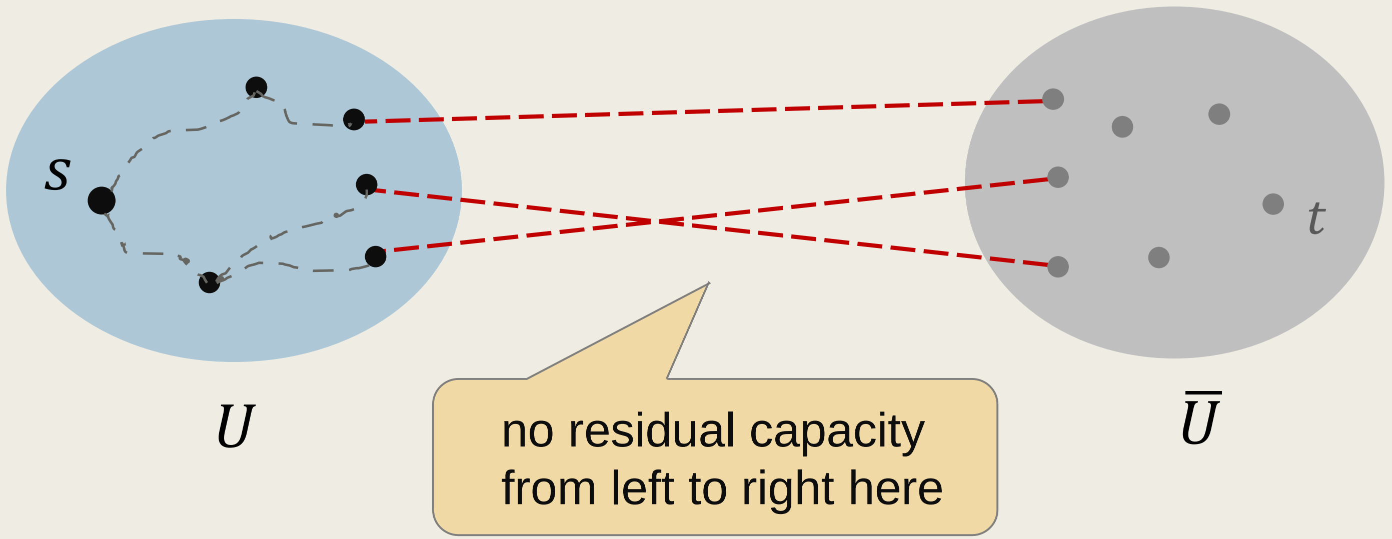
- Suppose that there exists no f-augmenting path in G_f.
- Let U be the set of vertices that are reachable from s via paths with positive residual capacity in G_f.
  - That is,

$$U = \{\, u \in V : \exists s - t \text{ path } P \text{ such that } c_f(a, b) > 0 \text{ for all } (a, b) \in P \,\}.$$



- Suppose that there exists no f-augmenting path in G_f.
- Let U be the set of vertices that are reachable from s via paths with positive residual capacity in G_f.
  - Let $\overline{U} = V \setminus U$.
  - Then, $c_f(a,b) = 0$, for all $(a,b) \in [U, \overline{U}]$.

$f(a,b) = c_{a,b}$ for all $(a,b) \in [U, \overline{U}]$.

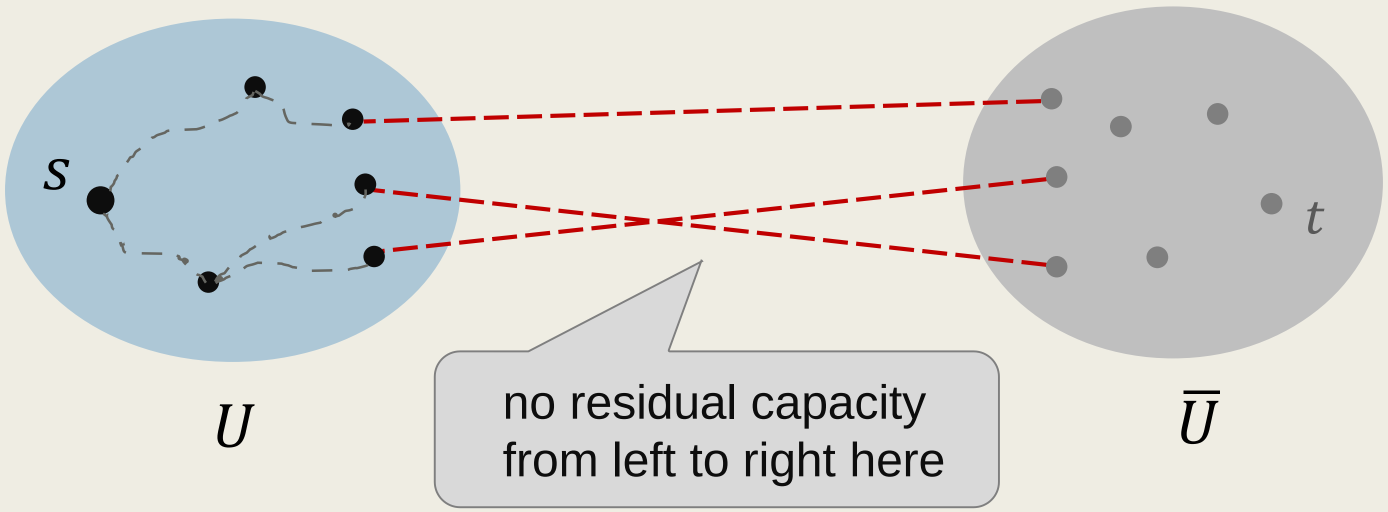


$f(a,b) = c_{a,b}$ for all $(a,b) \in [U, \overline{U}]$.

- Let $\overline{U} = V \setminus U$.
- Then, $c_f(a,b) = 0$, for all $(a,b) \in [U, \overline{U}]$.

- Hence,

$$\mathrm{val}(f) = f^+(U) = \sum_{\substack{a \in U \\ b \in \overline{U}}} f(a,b) = \sum_t c_{a,b} = w([U, \overline{U}]).$$



Time Complexity of the Ford-Fulkerson Algorithm

- In the worst-case,

  the Ford-Fulkerson algorithm takes $O(f \cdot (|V| + |E|))$ time.

  - Each flow augmentation takes $O(|V| + |E|)$ time to complete.

- The Ford-Fulkerson algorithm is not an efficient algorithm.

  - ***Its running time depends on the value of the input***,

    which can be exponential in the length of the input.

  - It is a pseudo-polynomial time algorithm.

Some Efficient Algorithms for Max-Flow

Efficient Algorithms for Max-Flow

- In the following,

  we sketch a few efficient algorithms for max-flow and min-cut.

  - The capacity scaling algorithm, $O\left(|E|^2 \cdot \log f\right)$.
  - The Edmonds-Karp algorithm, $O\left(|V| \cdot |E|^2\right)$.

Inspecting all the various flow algorithms is beyond the scope of this course.
Refer to concluding remarks for further references.

The Capacity Scaling Algorithm

- The capacity scaling algorithm works as follows.
  - Let Δ be the maximum capacity of the edges.
  - While $\Delta > 0$, do
    - Repeatedly compute f-augmenting path with value at least Δ in G_f and augment f by Δ until there is none.
    - Divide Δ by 2.
  - Output f.

The Capacity Scaling Algorithm

- It can be shown (by induction) that,
  - In each iteration,
    there are at most $O(|E|)$ f-augmenting paths with value $\geq \Delta$ in G_f.

- Hence, the total time complexity is $O\left(\log f \cdot |E|^2\right)$.

- Note that, in practice, $O(\log f) \ll O(|V|)$ almost always holds.

The Capacity Scaling Algorithm

- The capacity scaling algorithm is very easy to implement.
  - Almost as easy as the Ford-Fulkerson.
  - It takes less than 100 lines (with ample spacing and line-breaks) using only standard DFS.

The Edmonds-Karp Algorithm

- The Edmonds-Karp algorithm works as follows.
  - While there exists f-augmenting paths in G_f, do
    - Compute a shortest f-augmenting paths P, using BFS.
    - Use P to augment f by Δ_{P}.
  - Output f.

The Edmonds-Karp Algorithm

- It can be shown that,
  - The length of the shortest f-augmenting path between iterations is always nondecreasing and is at most $O(|V|)$.
  - The algorithm exhausts the capacity of at least one edge in each iteration.
    - The length of the shortest augmenting path will increase in $O(|E|)$ rounds.
- Hence, the total time complexity is $O\left(|V| \cdot |E|^2\right)$.

Concluding Notes

The Dinic's Algorithm

- The Dinic's algorithm is one of the best practical algorithm for max-flow.
  - It runs in $O\left(|V|^2 \cdot |E|\right)$.
  - It computes a maximal set of shortest augmenting paths in each iteration.
    - Similar to the Hopcroft-Karp algorithm for maximum bipartite matching.

Combining the Dinic's with the Capacity Scaling

- We can use Dinic's approach to compute a maximal set of f-augmenting paths with value Δ in G_f.
  - Then,
    the capacity scaling algorithm runs in $O(|V| \cdot |E| \cdot \log f)$.

The Best Algorithm for Max-Flow

- There are major breakthroughs in the max-flow problem in recent years.
- The best algorithm (so far) is given by the following research paper, which solves the max-flow problem in “almost linear time.”

Chen, Kying, Liu, Peng, Gutenberg, Sachdeva,
“Maximum Flow and Minimum-Cost Flow in Almost-Linear Time,”
arXiv:2203.00671, 2022.

This giant monster paper has 110 pages!!

- The best algorithm (so far) is given by the following research paper, which solves the max-flow problem in “almost linear time.”

Chen, Kying, Liu, Peng, Gutenberg, Sachdeva,
“Maximum Flow and Minimum-Cost Flow in Almost-Linear Time,”
arXiv:2203.00671, 2022.

  - It runs in $O\left(|E|^{1+o(1)}\right)$ time.
    The hidden constant, however, is very large.

  - It involves several complicated dynamic data structures.