Advanced Core in Algorithm Design #8
算法設計要論 第8回

Yasushi Kawase
河瀬 康志

Nov. 29th, 2022

last update: 3:27pm, November 29, 2022

Schedule

Lec. #	Date	Topics
1	10/4	Introduction, Stable matching
2	10/11	Basics of Algorithm Analysis, Greedy Algorithms (1/2)
3	10/18	Greedy Algorithms (2/2)
4	10/25	Divide and Conquer (1/2)
5	11/1	Divide and Conquer (2/2)
6	11/8	Dynamic Programming (1/2)
7	11/15	Dynamic Programming (2/2)
—	11/22	Thursday Classes
8	11/29	Network Flow (1/2)
9	12/6	Network Flow (2/2)
10	12/13	NP and Computational Intractability
11	12/20	Approximation Algorithms (1/2)
12	12/27	Approximation Algorithms (2/2)
13	1/10	Randomized Algorithms

Outline

1. Max-flow and Min-cut Problems
2. Augmenting path algorithm
3. Capacity-scaling algorithm

Flow Network

Flow network (G, s, t, c)

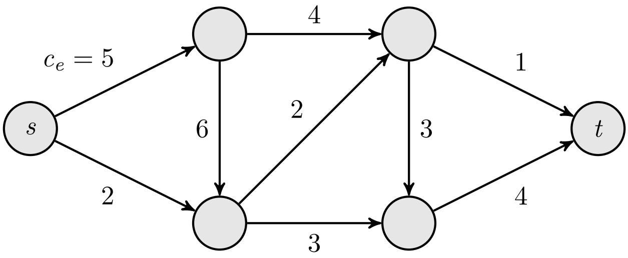
- directed graph $G = (V, E)$ with source $s \in V$ and sink $t \in V$
- capacity c_e for each $e \in E$

s–t flow f

Capacity constraint $0 \le f_e \le c_e$ for all $e \in E$

Conservation of flows $\sum_{u\colon e=(u,v)\in E} f_e = \sum_{u\colon (v,u)\in E} f_e$ $(\forall v \in V \setminus \{s, t\})$

Example



Flow Network

Flow network (G, s, t, c)

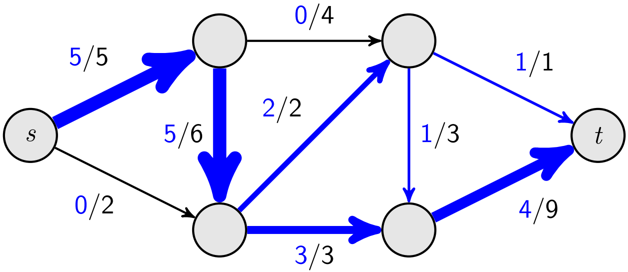
- directed graph $G = (V, E)$ with source $s \in V$ and sink $t \in V$
- capacity c_e for each $e \in E$

s–t flow f

Capacity constraint $0 \leq f_e \leq c_e$ for all $e \in E$

Conservation of flows $\sum_{u:\, e=(u,v)\in E} f_e = \sum_{u:\, (v,u)\in E} f_e$ $(\forall v \in V \setminus \{s, t\})$

Example



s–t flow

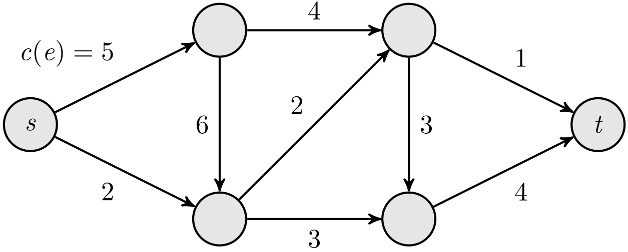
Max-flow problem

Problem

- Input: flow network (G, s, t, c)
- Goal: find an s–t flow of maximum value $\mathrm{val}(f)$

$\sum_{v:\, e=(s,v)\in E} f_e - \sum_{v:\, e=(v,s)\in E} f_e$

Example



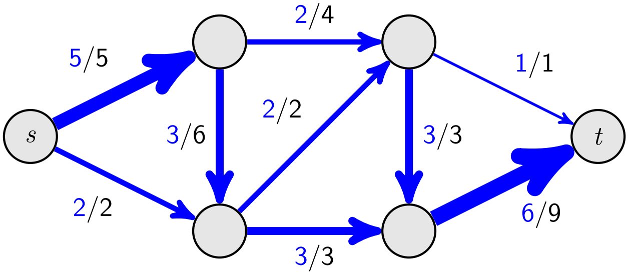
Max-flow problem

Problem

- Input: flow network (G, s, t, c)
- Goal: find an s–t flow of maximum value $\mathrm{val}(f)$

$\sum_{v:\, e=(s,v)\in E} f_e - \sum_{v:\, e=(v,s)\in E} f_e$

Example



$\mathrm{val}(f) = 7$

Min-cut Problem

s–t cut

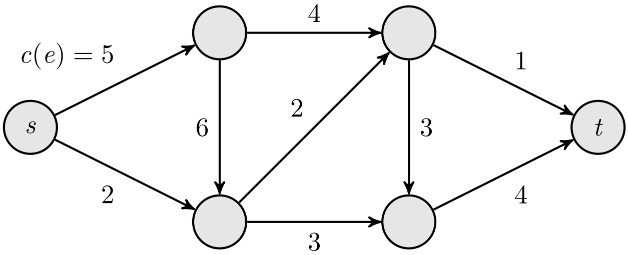
a partition (S, T) of the vertices with $s \in S$ and $t \in T$

Problem

- Input: flow network (G, s, t, c)
- Goal: find an s–t cut of minimum capacity $\mathrm{cap}(S)$

$\sum_{e=(u,v)\in E:\, u\in S,\, v\notin S} c_e$

Example



Min-cut Problem

s–t cut

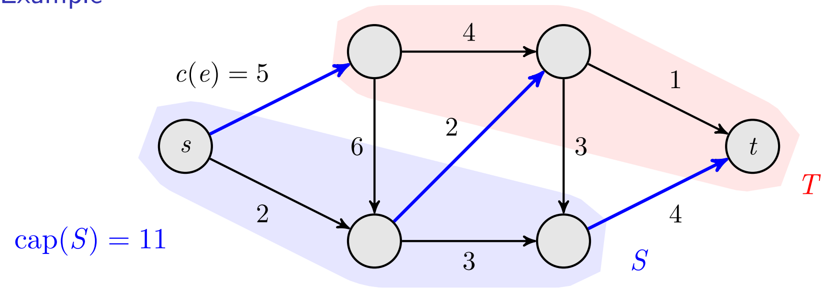
a partition (S, T) of the vertices with $s \in S$ and $t \in T$

Problem

- Input: flow network (G, s, t, c)
- Goal: find an s–t cut of minimum capacity $\mathrm{cap}(S)$

$\mathrm{cap}(S) = \sum_{e=(u,v)\in E:\, u\in S,\, v\notin S} c_e$

Example



Weak duality

Lemma

$\mathrm{val}(f) \leq \mathrm{cap}(S)$ for any flow f and cut (S, T)

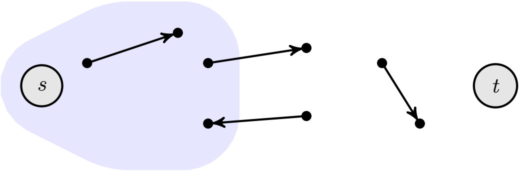
Proof

flow out of v — flow into v

$$\mathrm{val}(f) = \sum_{v \in S} \left[\sum_{u:\, e=(v,u)\in E} f_e - \sum_{u:\, e=(u,v)\in E} f_e \right]$$

$$= \sum_{e:\ \text{out of } S} f_e - \sum_{e:\ \text{into } S} f_e$$

$$\leq \sum_{e:\ \text{out of } S} f_e \leq \sum_{e:\ \text{out of } S} c_e = c(S)$$



Weak duality

Lemma

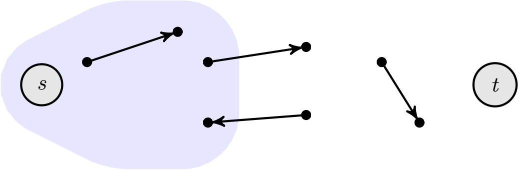
$\mathrm{val}(f) \le \mathrm{cap}(S)$ for any flow f and cut (S, T)

Proof

$$\mathrm{val}(f) = \sum_{v \in S} \left[\underbrace{\textstyle\sum_{u:\, e=(v,u)\in E} f_e}_{\text{flow out of } v} - \underbrace{\textstyle\sum_{u:\, e=(u,v)\in E} f_e}_{\text{flow into } v} \right]$$

$$= \sum_{e:\ \text{out of } S} f_e - \sum_{e:\ \text{into } S} f_e$$

$$\le \sum_{e:\ \text{out of } S} f_e \le \sum_{e:\ \text{out of } S} c_e = c(S)$$



we will see $\max_f \mathrm{val}(f) = \min_{(S,T)} \mathrm{cap}(S)$

Outline

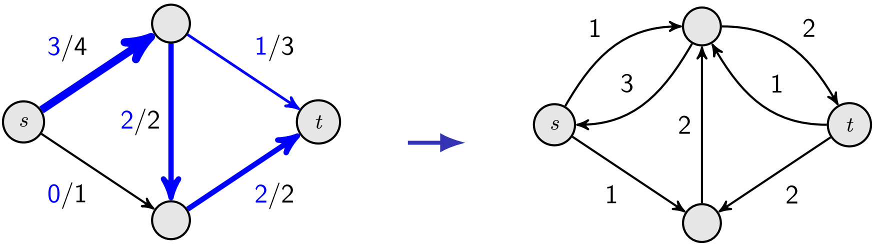
1. Max-flow and Min-cut Problems
2. Augmenting path algorithm
3. Capacity-scaling algorithm

Residual network (G_f, s, t, c_f) of G w.r.t. f

reverse edge of e

residual graph $G_f = (V, E_f)$, $E_f = \{e \mid e \in E,\ f_e < c_e\} \cup \{\bar{e} \mid e \in E,\ f_e > 0\}$

residual capacity $c_f(e) = \begin{cases} c_e - f_e & \text{if } e \in E \\ f_e & \text{if } \bar{e} \in E \end{cases}$

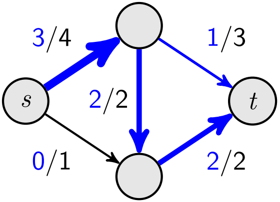


original network and flow → residual network

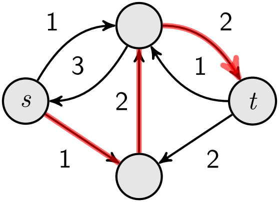
Augmenting path

- augmenting path: a simple s–t path in the residual network G_f
- bottleneck capacity: the minimum residual capacity of a path

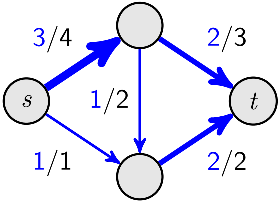
bottleneck capacity $= 1$



original network and flow



residual network and augmenting path



new flow

Ford–Fulkerson algorithm

$\mathrm{Augment}(f, c, P)$

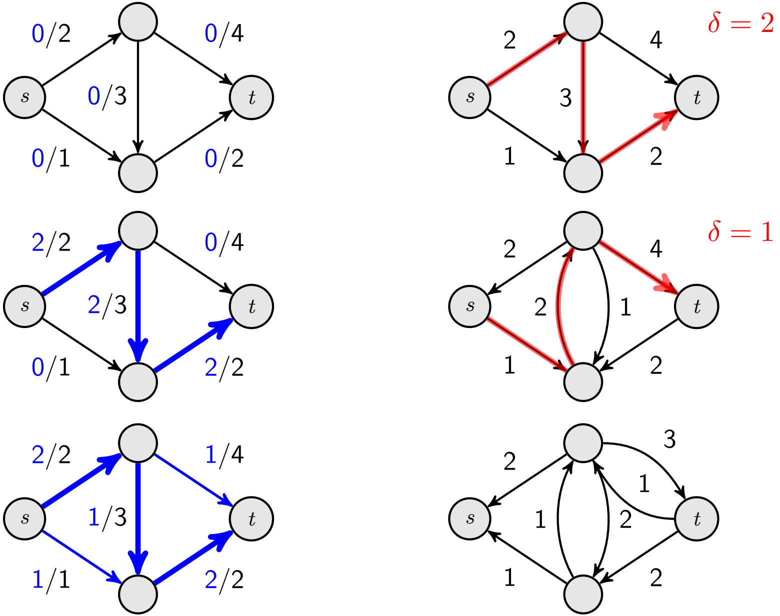
$\delta \leftarrow$bottleneck capacity of augmenting path P;
foreach $e \in P$ **do**
  if $e \in E$ **then** $f_e \leftarrow f_e + \delta$;
  else $f_{\bar{e}} \leftarrow f_{\bar{e}} - \delta$;
Return f;

Output of $\mathrm{Augment}(f, c, P)$ is a flow

Ford–Fulkerson algorithm

$f_e \leftarrow 0$ for each $e \in E$;
$G_f \leftarrow$ residual network of G with respect to flow f;
while $\exists$ *an s–t path P in G_f* **do**
  $f \leftarrow \mathrm{Augment}(f, c, P)$;
  Update G_f;

Example


0/2
0/4
s
0/3
t
0/1
0/2
δ = 2
2
4
s
3
t
1
2
2/2
0/4
s
2/3
t
0/1
2/2
δ = 1
2
4
s
2
1
t
1
2
2/2
1/4
s
1/3
t
1/1
2/2
2
3
1
s
1
2
t
1
2

Termination and Running time

Suppose that $c_e \in \mathbb{Z}_{++}$ $(\forall e \in E)$

Observation

$\forall$ iterations, the flow value f_e and the residual capacity of G_f are integral

Observation

$\forall$ iterations, $\mathrm{val}(f)$ increases at least 1

Theorem

- Ford–Fulkerson algorithm terminates in at most $C := \sum_{e \in E} c_e$ steps
- Ford–Fulkerson algorithm can be implemented to run in $\mathrm{O}(mC)$ time

s–t path can be found in $\mathrm{O}(m)$ time by BFS or DFS

Correctness

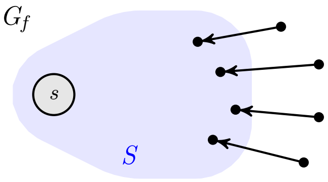
Theorem

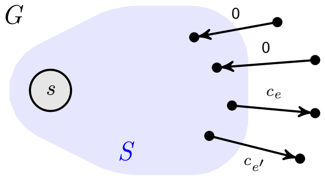
Ford–Fulkerson algorithm outputs a max-flow

Proof

- When it terminates, $\nexists$ s–t path in G_f
- Let S be the set of vertices reachable from s in G_f ($s \in S$ and $t \notin S$)
- $\mathrm{val}(f) = \sum_{e\text{: out of } S} f_e - \sum_{e\text{: into } S} f_e = \sum_{e\text{: out of } S} c_e = \mathrm{cap}(S)$
- By the weak duality, f is a max-flow and $(S, V \setminus S)$ is a min-cut

$\mathrm{val}(f') \leq \mathrm{cap}(A)$



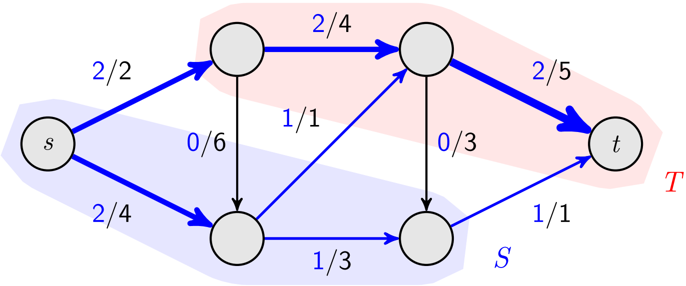


Max-flow Min-cut theorem

Theorem

$\max_{f:\,\text{flow}} \operatorname{val}(f) = \min_{(S,T):\text{cut}} \operatorname{cap}(S)$

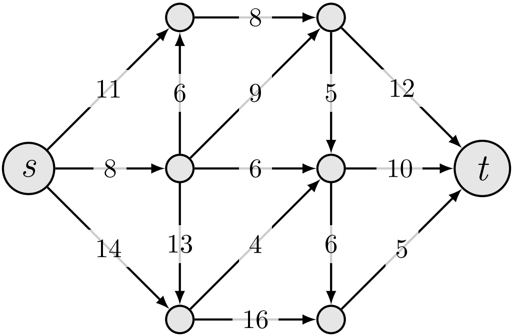
Example



$\operatorname{val}(f) = \operatorname{cap}(S) = 4$

Quiz

What is the maximum value of s–t flow?

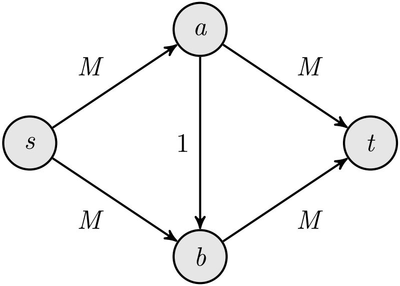


Outline

1. Max-flow and Min-cut Problems
2. Augmenting path algorithm
3. Capacity-scaling algorithm

Bad example

Ford–Fulkerson is too slow (exponential time w.r.t. input size)



- $s \to a \to b \to t$
- $s \to b \to a \to t$
- $s \to a \to b \to t$
- $s \to b \to a \to t$
- $\vdots$

we'd like to choose "good" augmenting path

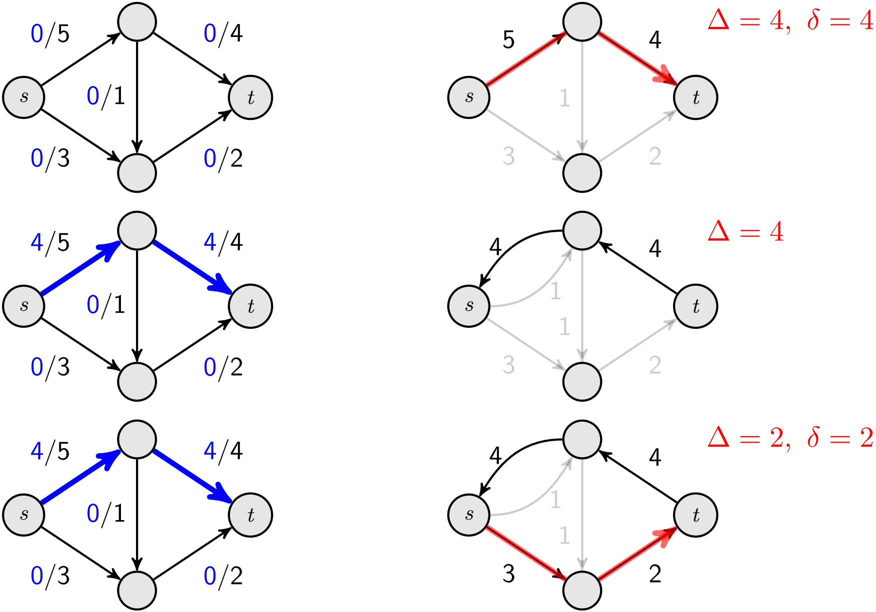
Capacity scaling

- Choosing augmenting paths with large bottleneck capacity
- $G_f(\Delta)$: subgraph of G_f consisting only of edges e with $c_f(e) \geq \Delta$

Scaling algorithm

```
Δ ← largest power of 2 that is no larger than max_{e: out of s} c_e;
G_f ← residual network of G with respect to flow f;
while Δ ≥ 1 do
    while ∃ an s–t path P in G_f(Δ) do
        f ← Augment(f, c, P);
        Update G_f(Δ);
    Δ ← Δ/2;
```

Example



Analyzing the algorithm

Lemma

$\max_{e:\text{ out of } s} c_e$

The number of scaling phases is $1 + \lfloor \log_2 C \rfloor$

$\because\ \Delta = 2^{\lfloor \log_2 C \rfloor}, 2^{\lfloor \log_2 C \rfloor - 1}, \ldots, 2^0$

Lemma

the set of vertices reachable from s in $G_f(\Delta)$

At the end of a Δ-scaling phase, $\mathrm{cap}(S) \le \mathrm{val}(f) + m\Delta$

see next slide

Lemma

The number of augmentations in each scaling phase is at most $2m$

- at the beginning of Δ-scaling phase, max-flow$\le \mathrm{val}(f) + m(2\Delta)$
- each augmentation increases $\mathrm{val}(f)$ by at least Δ

Theorem

weakly polynomial time

The scaling algorithm can be implemented to run in $\mathrm{O}(m^2 \log C)$ time

finding an augmenting path takes $\mathrm{O}(m)$ time

Proof of the lemma

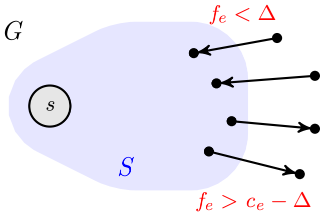
Lemma

the set of vertices reachable from s in $G_f(\Delta)$

At the end of a Δ-scaling phase, $\mathrm{cap}(S) \le \mathrm{val}(f) + m\Delta$

Proof

$$\begin{aligned}\mathrm{val}(f) &= \sum_{e:\ \text{out of } S} f_e - \sum_{e:\ \text{into } S} f_e \\ &\ge \sum_{e:\ \text{out of } S} (c_e - \Delta) - \sum_{e:\ \text{into } S} \Delta \\ &\ge \mathrm{cap}(S) - m\Delta\end{aligned}$$



Summary

- Ford–Fulkerson algorithm ⟶ $\mathrm{O}(mC)$ time (pseudo polynomial-time)
- Scaling algorithm ⟶ $\mathrm{O}(m^2 \log C)$ time (weak polynomial-time)

Next week

- Edmonds–Karp algorithm ⟶ $\mathrm{O}(m^2 n)$ time (strong polynomial-time)

State of the Art

- $\mathrm{O}(mn)$ time [Orlin 2013]
- $m^{1+o(1)} \log C$ time [Chen et al., 2022]