Lecture 9

Connectivity: Dijkstra's algorithm.
Flow networks: Maximum flow algorithms

December, 7 2015

Lecture outline

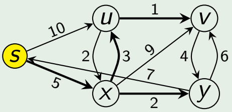
1. The problem of lightest paths from a single source in a weighted digraph
    - **Dijkstra's algorithm**
2. Flow networks and flows
    - Maximum flow
    - Residual networks, augmenting paths
    - **Ford-Fulkerson algorithm**
    - Applications

Lightest paths from a given source node

Given a simple weighted digraph $G = (V, E)$ with $w : E \mapsto \mathbb{R}^+$ and a source node $s \in V$

Find for every node $x \in V$ accessible from s, a lightest path $\rho : s \rightsquigarrow x$, and its weight $w(\rho)$

Example



$[s]$ with $w([s]) = 0$; $[s, x, u]$ with $w([s, x, u]) = 8$

$[s, x]$ with $w([s, x]) = 5$; $[s, x, u, v]$ with $w([s, x, u, v]) = 9$

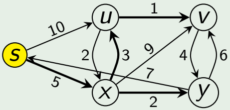
$[s, x, y]$ with $w([s, x, y]) = 7$.

Lightest paths from a given source node

Given a simple weighted digraph $G = (V, E)$ with $w : E \mapsto \mathbb{R}^+$ and a source node $s \in V$

Find for every node $x \in V$ accessible from s, a lightest path $\rho : s \rightsquigarrow x$, and its weight $w(\rho)$

Example



$[s]$ with $w([s]) = 0$; $[s, x, u]$ with $w([s, x, u]) = 8$

$[s, x]$ with $w([s, x]) = 5$; $[s, x, u, v]$ with $w([s, x, u, v]) = 9$

$[s, x, y]$ with $w([s, x, y]) = 7$.

Remark

- The problem can be solved with Warshall's algorithm:
  - Computes the lightest paths that exist between every pair of nodes
  - Runtime complexity $O(|V|^3)$; it computes more than needed

Is there a better algorithm, if the source node is fixed?

Proposed by E. Dijkstra in 1956 to solve the previous problem

1. Assign
   - A tentative weight $d(x)$ for a lightest path from source to x.
   - a predecessor node $\pi(x)$ of every node x on a lightest path from s to x.

   Initially, we have $d(x) = \begin{cases} 0 & \text{if } x = s, \\ \infty & \text{if } x \neq s \end{cases}$ $\quad \pi(x) = \begin{cases} \mathit{undef} & \text{if } x = s \\ s & \text{if } x \neq s \end{cases}$
   where *undef* is a special value: it indicates the inexistence of a predecessor.
2. Create a set Q of unvisited nodes. Initially, $Q := V$, and keep track of a current node crt.
3. choose crt :=a node form Q with $d(crt) = \min\{d(x) \mid x \in Q\}$, and remove crt from Q.
4. For every neighbor $x \in Q$ of crt update the tentative values of $d(x)$ and $\pi(x)$ as follows:

   If $d(crt) + w((crt, x)) < d(x)$ then $d(x) := d(crt) + w((crt, x))$ and $\pi(x) := crt$.

   This updating step is called relaxation step of the arc $(crt, x) \in E$.
5. If $Q = \emptyset$ then **stop**, else **goto 3**.

- Initialization

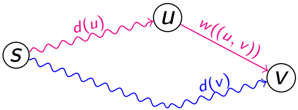
  $\textsc{SingleSourceInit}(G, s)$

  for each $v \in V$
  $d(v) := \infty$
  $\pi(v) := s$
  $d(s) := 0$
  $\pi(s) := undef$

- Relaxation step for an arc (u, v)

  $\textsc{Relax}(u, v)$

  if $d(v) > d(u) + w((u, v))$
  $d(v) := d(u) + w((u, v))$
  $\pi(v) := \pi(u)$



Dijkstra's algorithm
Pseudocode

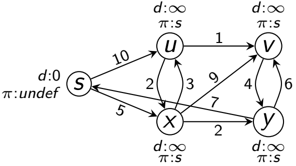
```
DIJKSTRA(G, w, s)
SINGLESOURCEINIT(G, s)
Q := V
while Q ≠ ∅
    u := EXTRACTMIN(Q)
    for every neighbor v of u for which v ∉ Q
        RELAX(u, v)
```

Runtime complexity:

- Original algorithm: $O(|V|^2)$
- Algorithm improved with a min-priority queue: $O(|E| + |V| \cdot \log |V|)$

Dijkstra's algorithm
Illustrated example: first **while** loop

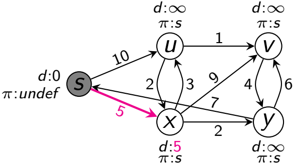
CONVENTION: The nodes not marked yet (those from Q) are white; the others are gray

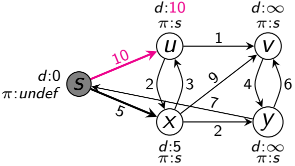


Configuration produced by INITIALIZESINGLESOURCE(G, s):

$Q = \{s, x, y, u, v\}$
Select s = EXTRACTMIN(Q)

Relax all arcs from s to nodes not visited yet:

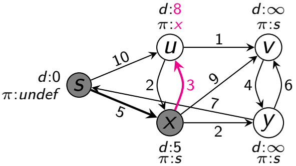


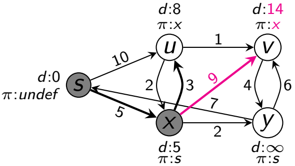


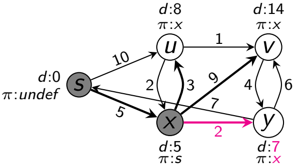
Dijkstra's algorithm

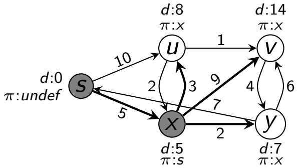
Illustrated example: the second **while** loop

Select and mark x, and relax all arcs from x to unmarked nodes:

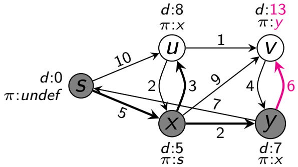






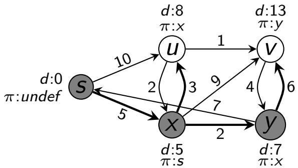


Select and mark y, and relax all arcs from y to unmarked nodes:

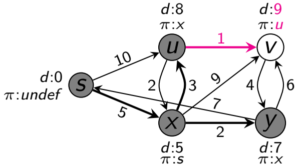


Dijkstra's algorithm

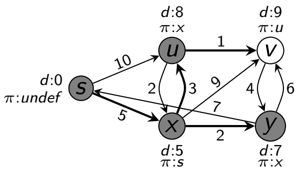
Illustrated example: the fourth **while** loop



Select and mark u, and relax all arcs from u to unmarked nodes:



Dijkstra's algorithm
Illustrated example: the fifth **while** loop



$d(s) = 0$ $\pi(s) = undef$

$d(x) = 5$ $\pi(x) = s$

$d(u) = 8$ $\pi(u) = x$

$d(y) = 7$ $\pi(y) = x$

$d(v) = 9$ $\pi(v) = u$

- Select and mark v
- There are no arcs left to relax $\Rightarrow$ the algorithm stops.

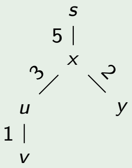
From the values of π and d we can retrieve lightest paths from s to all other nodes:

- to s: $[s]$ with weight $w([s]) = d(s) = 0$
- to x: $[s, x]$ with weight $w([s, x]) = d(x) = 5$
- to u: $[s, x, u]$ with weight $w([s, x, u]) = d(u) = 8$
- to y: $[s, x, y]$ with weight $w([s, x, y]) = d(y) = 7$
- to v: $[s, x, u, v]$ with weight $w([s, x, u, v]) = d(v) = 9$

The tree of lightest paths form source to all other nodes

The function π computed by Dijkstra's algorithm determines a tree G_π with root s, in which every node $x \neq s$ has parent $\pi(x)$.

Example (The tree G_π for the illustrated weighted digraph G)



Remark

Every branch of G_π from the source node s to a node x is a lightest path from s to x.

References

1. T. H. Cormen, C. E. Leiserson, R. L. Rivest. Section **25**.2 from *Introduction to Algorithms*. MIT Press, 2000.
2. A `C++` implementation of Dijkstra's algorithm can be downloaded from the website of this lecture (click here)

Flow network: Oriented graph in which arch represent flows of material between nodes (volume of liquid, electricity, a.s.o.)

- Every edge has a maximum capacity.
- We wish to determine a flow from a source node (the **producer**) to a sink node (the **consumer**).

Flow $\approx$ the rate of flow of resources along arcs .

The problem of maximum flow: What is the maximum possible flow of resources from source to destination, without violating any maximum capacity constraint of the arcs?

Definition (Flow network)

An oriented graph $G = (V, E)$, where every arc $(u, v) \in E$ has a capacity $c(u, v) \geq 0$, and two special nodes:

- a source s and
- a sink t.

If $(u, v) \notin E$, we assume $c(u, v) = 0$.
We write $u \rightsquigarrow v$ to indicate the existence of a path from u to v, and assume that every node $v \in G$ is on a path from s to t, i.e., there is a path $s \rightsquigarrow v \rightsquigarrow t$.

Remark

A flow network is a connected graph, thus $|E| \geq |V| - 1$.

Flows

Definition

A flow in a flow network G is a function $f : V \times V \to \mathbb{R}$ that fulfils the following constraints:

Capacity constraint: For all $u, v \in V$, $f(u, v) \leq c(u, v)$.

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

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

$f(u, v)$ is called the net flow from node u to v. The value of a flow f is defined as $|f| = \sum_{v \in V} f(s, v)$, that is, the total net flow out of the source.

The maximum-flow problem

Given a flow network G

Find a flow of maximum value from s to t.

- The positive net flow entering a node v is

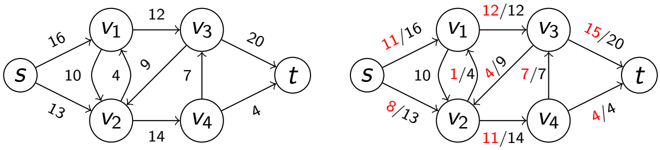
$$\sum_{\substack{u \in V \\ f(u,v)>0}} f(u,v)$$

- The positive net flow leaving a node v is

$$\sum_{\substack{u \in V \\ f(v,u)>0}} f(v,u)$$

$\Rightarrow$ by flow conservation property: for all nodes v, the positive net flow entering node v = the positive net flow leaving node v.

Network flow example



(a) (b)

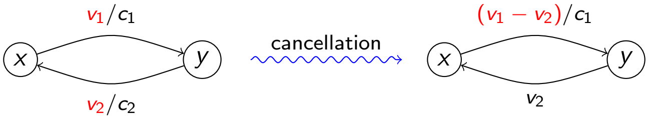
(a) A flow network $G = (V, E)$ with edges labeled with their capacities. The source is s, and destination is t.

(b) A flow f in the flow network G with value $|f| = 19$. Only positive flows are shown. If $f(u, v) > 0$, edge (u, v) is labeled with $f(u, v)/c(u, v)$. (The slash notation is used merely to separate the flow and capacity; it does *not* indicate division.) If $f(u, v) \leq 0$, edge (u, v) is labeled only by its capacity.

Network flows

Removing all negative net flows – the cancelation rule

If $v_1 \geq v_2$ then

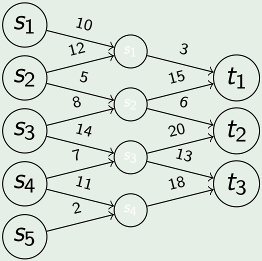


- Only positive net flows represent actual shipments.
- Applications of the cancelation rule
  - eliminate negative net flows.
  - do not violate the 3 requirements of a network flow:
    1. capacity constraint
    2. skew symmetry
    3. flow conservation

Multiple sources and sinks

- A maximum-flow problem can have several sources $s_1, \ldots, s_m$ and sinks $t_1, \ldots, t_m$.
- Such a problem can be reduced to an equivalent single-source single-sink maximum-flow problem:
  - add a **supersource** s and a **supersink** t
  - add directed edges (s, s_i) with $c(s, s_i) = \infty$ for $i = 1..m$
  - add directed edges (t_j, t) with $c(t_j, t) = \infty$ for $j = 1..n$

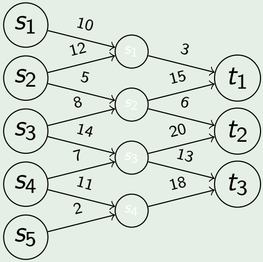
Example

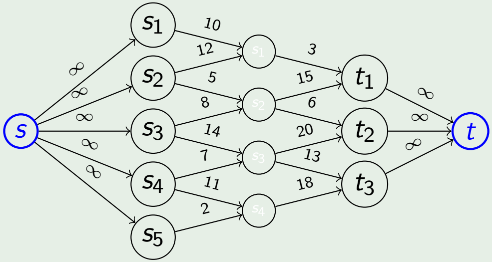


Multiple sources and sinks

- A maximum-flow problem can have several sources $s_1, \ldots, s_m$ and sinks $t_1, \ldots, t_m$.
- Such a problem can be reduced to an equivalent single-source single-sink maximum-flow problem:
  - add a **supersource** s and a **supersink** t
  - add directed edges (s, s_i) with $c(s, s_i) = \infty$ for $i = 1..m$
  - add directed edges (t_j, t) with $c(t_j, t) = \infty$ for $j = 1..n$

Example





- Assume we know:

  a flow network $G = (V, E)$
  a function f from $V \times V$ to $\mathbb{R}$
  sets of nodes X, Y (that is, $X \subseteq V$, $Y \subseteq V$)
  node $u \in V$.

- Then
  - $f(X, Y)$ represents the sum $\displaystyle\sum_{x \in X} \sum_{y \in Y} f(x, y)$.
  - $f(u, X)$ represents the sum $\displaystyle\sum_{x \in X} f(u, x)$.
  - $f(Y, u)$ represents the sum $\displaystyle\sum_{y \in Y} f(y, u)$.
  - $X - u$ represents the set $X - \{u\}$.

Remark. If f is a flow for $G = (V, E)$ then $f(u, V) = 0$ for all $u \in V - \{s, t\}$. This follows from the flow conservation constraint $\Rightarrow f(V - \{s, t\}, V) = 0$.

Properties of flow networks

Lemma

Let $G = (V, E)$ be a flow network and f a flow in G. Then

- $f(X, X) = 0$ for all $X \subseteq V$.
- $f(X, Y) = -f(Y, X)$ for all $X, Y \subseteq V$.
- $f(X \cup Y, Z) = f(X, Z) + f(Y, Z)$ and $f(Z, X \cup Y) = f(Z, X) + f(Z, Y)$ for all $X, Y, Z \subseteq V$ with $X \cap Y = \emptyset$.

Note that:

$$
\begin{aligned}
|f| &= f(s, V) && \text{by definition} \\
&= f(V, V) - f(V - s, V) && \text{by previous lemma} \\
&= f(V, V - s) && \text{by previous lemma} \\
&= f(V, t) + f(V, V - \{s, t\}) && \text{by previous lemma} \\
&= f(V, t) && \text{by flow conservation}
\end{aligned}
$$

Operations with flows

Definition

If f_1, f_2 are flows in a flow network G and $\alpha \in \mathbb{R}$, then

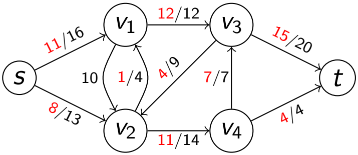
- the flow sum $f_1 + f_2$ of f_1 and f_2 is the function from $V \times V$ to $\mathbb{R}$ defined by

$$(f_1 + f_2)(u, v) := f_1(u, v) + f_2(u, v) \quad \text{for all } u, v \in V.$$

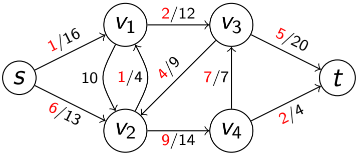
- the scalar flow product αf_1 is the function from $V \times V$ to $\mathbb{R}$ defined by

$$(\alpha f_1)(u, v) := \alpha f_1(u, v) \quad \text{for all } u, v \in V.$$

Operations with flows
Examples



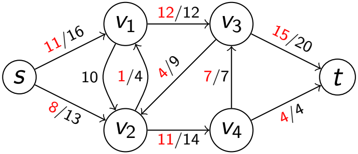
(a) G and f_1



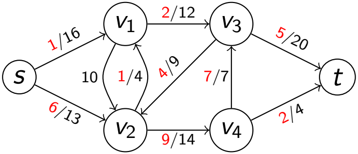
(b) G and f_2

Operations with flows

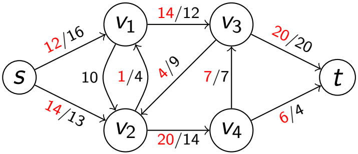
Examples



(a) G and f_1



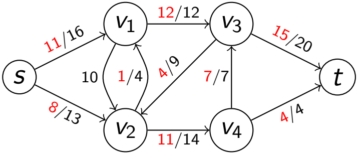
(b) G and f_2



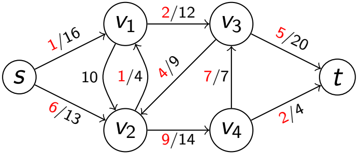
(c) G and $f_1 + f_2$

Operations with flows

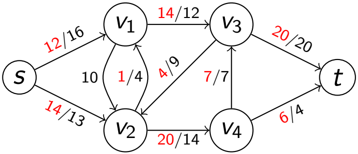
Examples



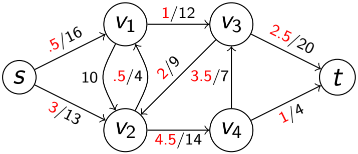
(a) G and f_1



(b) G and f_2



(c) G and $f_1 + f_2$



(d) G and αf_2 when $\alpha = \frac{1}{2}$

A flow must satisfy 3 requirements: capacity constraint, skew symmetry, and flow conservation.

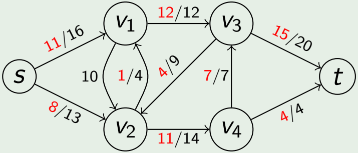
1. Which properties are not preserved by flow sums?
2. Which properties are not preserved by scalar flow products?
3. Show that, if f_1, f_2 are flows and $0 \leq \alpha \leq 1$, then $\alpha f_1 + (1-\alpha) f_2$ is a flow.

Residual networks

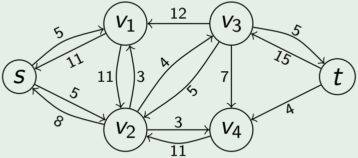
Assumptions: a flow network $G = (V, E)$; flow f in G.

- The residual capacity of an edge (u, v) is $c_f(u, v) := c(u, v) - f(u, v)$.
- The residual network of G induced by f is the flow network $G_f = (V, E_f)$ where $E_f = \{(u, v) \in V \times V \mid c_f(u, v) > 0\}$, and the capacity of every edge is (u, v) is $c_f(u, v)$.

Example



(a) G and f



(b) G_f

Remark. In general, $|E_f| \leq 2|E|$.

Assume a flow network G, a flow f in G, and the residual network G_f. If f' is a flow in G_f then $f+f'$ is a flow in G with value $|f+f'| = |f| + |f'|$.

PROOF.

- **Skew symmetry** holds because $(f+f')(u,v) = f(u,v) + f'(u,v) = -f(v,u) - f'(v,u) = -(f(v,u) + f'(v,u)) = -(f+f')(v,u)$.
- For the **capacity constraints**, note that $f'(u,v) \leq c_f(u,v)$ for all $u,v \in V$, therefore $(f+f')(u,v) = f(u,v) + f'(u,v) \leq f(u,v) + (c(u,v) - f(u,v)) = c(u,v)$.
- For **flow conservation**, we note that

$$\begin{aligned}\sum_{v\in V}(f+f')(u,v) &= \sum_{v\in V}(f(u,v)+f'(u,v))\\ &= \sum_{v\in V} f(u,v) + \sum_{v\in V} f'(u,v) = 0+0 = 0.\end{aligned}$$

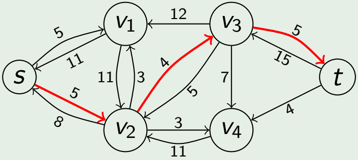
Finally, we have

$$|f+f'| = \sum_{v\in V}(f+f')(s,v) = \sum_{v\in V}(f(s,v)+f'(s,v)) = \sum_{v\in V} f(s,v) + \sum_{v\in V} f'(s,v) = |f| + |f'|.$$

Augmenting paths

An augmenting path for a flow network G and a flow f is a simple path from s to t in the residual network G_f.

Example (Augmented path)



REMARKS.

- Each edge (u, v) of an augmenting path admits additional positive net flow without violating the capacity of the edge.
- In this example, we could ship up to 4 units more from s to t along the highlighted augmenting path, without violating any capacity constraint (Note: the smallest residual capacity on the highlighted augmenting path is 4).

Augmenting paths (continued)

- The residual capacity of an augmenting path p is given by

$$c_f(p) := \min\{c_f(u,v) \mid (u,v) \text{ is on } p\}.$$

Lemma

Let $G = (V, E)$ be a flow network with flow f, p an augmenting path in G_f, and $f_p : V \times V \to \mathbb{R}$ defined by

$$f_p(u,v) := \begin{cases} c_f(p) & \text{if } (u,v) \text{ is on } p, \\ -c_f(p) & \text{if } (v,u) \text{ is on } p, \\ 0 & \text{otherwise.} \end{cases}$$

Then f_p is a flow in G_f with value $|f_p| = c_f(p) > 0$.

Corollary

Let $G = (V, E)$ be a flow network with flow f, and p be an augmenting path in G_f. Let f_p be the flow defined as in the previous lemma. Then $f + f_p$ is a flow in G with value $|f'| = |f| + |f_p| > |f|$.

The Ford-Fulkerson method

Yields a maximum flow for a given flow network G:

```
FORD-FULKERSON-METHOD(G, s, t)
initialize flow f to 0
while there exists an augmenting path p
        augment flow f along p
return f
```

- The Ford-Fulkerson method works because the following result holds:

  A flow is maximum if and only if its residual network contains no augmenting path.

The Ford-Fulkerson method

Yields a maximum flow for a given flow network G:

Ford-Fulkerson-Method(G, s, t)
1 initialize flow f to 0
2 **while** there exists an augmenting path p
3 augment flow f along p
4 **return** f

- The Ford-Fulkerson method works because the following result holds:

  A flow is maximum if and only if its residual network contains no augmenting path.

▷ We shall prove this fact.

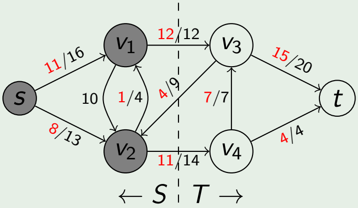
Auxiliary notions: cut, capacity of a cut.

Cuts

Definition

A cut (S, T) of a flow network $G = (V, E)$ is a partition of V into S and $T = V - S$ such that $s \in S$ and $t \in T$. The net flow across the cut (S, T) is $f(S, T)$. The capacity of the cut (S, T) is $c(S, T)$.

Example



$S = \{s, v_1, v_2\}$
$T = \{v_3, v_4, t\}$

$f(S, T) = f(v_1, v_3) + f(v_2, v_3) + f(v_2, v_4) = 12 + (-4) + 11 = 19$
$c(S, T) = c(v_1, v_3) + c(v_2, v_4) = 12 + 14 = 26$

Lemma

The net flow across a cut (S, T) if $f(S, T) = |f|$.

Corollary

For any flow f and any cut (S, T), we have $|f| \leq c(S, T)$.

Max-flow min-cut theorem

If f is a flow in a flow network $G = (V, E)$ with source s and sink t, then the following conditions are equivalent:

1. f is a maximum flow in G.
2. G_f contains no augmenting paths.
3. $|f| = c(S, T)$ for some cut (S, T) of G.

$(1) \Rightarrow (2)$ By contradiction: Assume f is a maximum flow in G and that G_f has an augmenting path p. Then $f + f_p$ would be a flow in G with value strictly larger than $|f|$, contradicting the assumptions.

$(2) \Rightarrow (3)$ Suppose G_f has no augmenting path from s to t. Let

$$S = \{v \in V \mid \text{there exists a path from } s \text{ to } v \text{ in } G_f\}$$

and $T = V - S$. Then (S, T) is a cut because $s \in S$ and $t \notin S$. For each pair of nodes $(u, v) \in S \times T$ we have $v(u, v) = c(u, v)$ because otherwise $(u, v) \in E_f$ and $v \in S$. It follows that $|f| = f(S, T) = c(S, T)$.

$(3) \Rightarrow (1)$ We know that $|f| \leq c(S, T)$ for all cuts (S, T) of G. Therefore, the condition $|f| = c(S, T)$ implies that f is a maximum flow.

Assume

1. $G = (V, E)$ is a flow network,
2. f is a maximum flow in G,
3. (S, T) is a cut of G with minimum capacity.

Then

- $|f| = c(S', T')$ for some cut (S', T') of G. Since $c(S, T) \leq c(S', T')$ (by assumption 3), we have $c(S, T) \leq |f|$.
- By Previous corollary, $|f| \leq$ capacity of any cut; in particular $|f| \leq |c(S, T)|$.

$\Rightarrow |f| = c(S, T)$. This means that

$\triangleright$ Value of maximum flow in G = minimum capacity of cut of G.

The basic Ford-Fulkerson algorithm

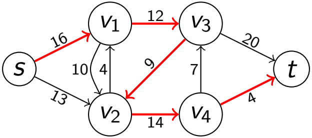
```
Ford-Fulkerson(G, s, t)
for each edge (u, v) ∈ E(G)
    f(u, v) := 0
    f(v, u) := 0
while ∃ path p from s to t in G_f
    c_f := min{c_f(u, v) | (u, v) is in p}
    for each edge (u, v) in p
        f(u, v) := f(u, v) + c_f(p)
        f(v, u) := −f(u, v)
```

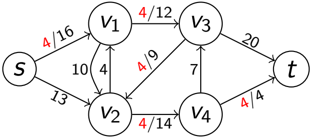
The basic Ford-Fulkerson algorithm
Running example

Residual network G_f with augmented path (line 4)

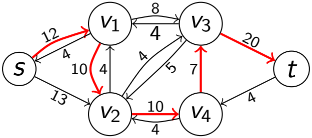
New flow that results from adding f_p to f

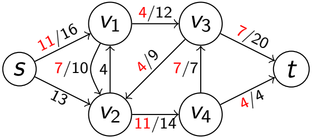
(a)



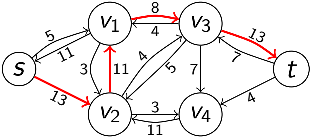


(b)





(c)



... ...

Exercise: draw the graphs for the remaining steps of Ford-Fulkerson algorithm.

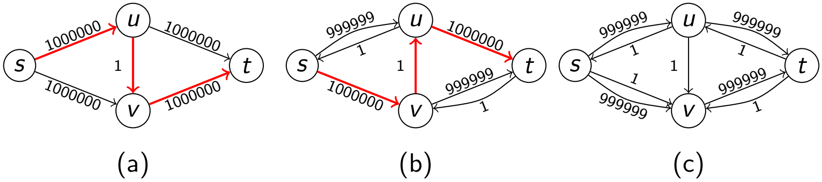
- The running time depends on how the augmenting path p is computed in line 4 of the algorithm.

- The running time depends on how the augmenting path p is computed in line 4 of the algorithm.
- ASSUMPTION: all edge capacities are integral numbers (that is, 0,1,2,. . .).

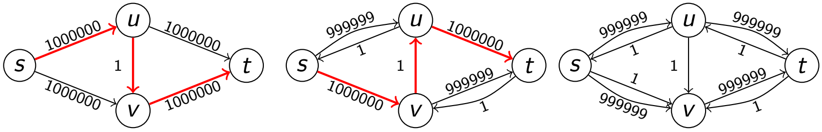
- The running time depends on how the augmenting path p is computed in line 4 of the algorithm.
- Assumption: all edge capacities are integral numbers (that is, 0,1,2,...).
  - If the capacities are rational numbers, we can make them all integer, with an appropriate scaling transformation.

- The running time depends on how the augmenting path p is computed in line 4 of the algorithm.
- ASSUMPTION: all edge capacities are integral numbers (that is, 0,1,2,...).
  - If the capacities are rational numbers, we can make them all integer, with an appropriate scaling transformation.
- A straightforward implementation of FORD-FULKERSON algorithm runs in time $O(|E| \cdot |f^*|)$ where f^* is the maximum flow found by the algorithm.

  Reason: the **while** loop of lines 4-8 is executed at most $|f^*|$ times, because the flow values increase by at least 1 in each iteration.



- A maximum flow f^* in flow network (a) has $|f^*| = 2000000$. A poorly chosen augmented path, with capacity 1, is highlighted.
- (b) and (c) illustrate resulting residual networks, after augmenting with the previously highlighted augmenting path.



(a) (b) (c)

- A maximum flow f^* in flow network (a) has $|f^*| = 2000000$. A poorly chosen augmented path, with capacity 1, is highlighted.
- (b) and (c) illustrate resulting residual networks, after augmenting with the previously highlighted augmenting path.
- Time complexity is improved if p in line 4 is computed with a breadth-first search, that is, if p is a *shortest* path from s to t in the residual network, where each edge has unit distance (weight) $\Rightarrow$ Edmonds-Karp algorithm with runtime complexity $O(|V| \cdot |E|^2)$.

Let $B = (V_1 \cup V1, E)$ be a bipartite graph between subsets V_1 and V_2 of V (Note: $V_1 \cap V_2 = \emptyset$.)

Definition

A matching in B is a set of edges $M \subseteq E$ such that for all nodes v of G, at most one edge of M is incident on v. A maximum matching is a matching of maximum cardinality, that is, a matching M such that for any matching M', we have $|M| \geq |M'|$.

Let $B = (V_1 \cup V1, E)$ be a bipartite graph between subsets V_1 and V_2 of V (Note: $V_1 \cap V_2 = \emptyset$.)

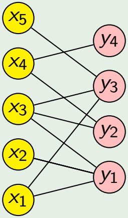
Definition

A matching in B is a set of edges $M \subseteq E$ such that for all nodes v of G, at most one edge of M is incident on v. A maximum matching is a matching of maximum cardinality, that is, a matching M such that for any matching M', we have $|M| \geq |M'|$.

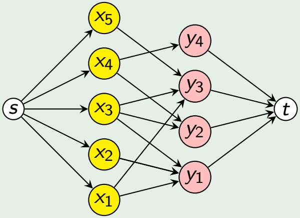
A maximum bipartite matching of $B = (V_1 \cup V_2, E)$ can be found as follows:

1. Extend B with 2 new nodes: s (supersource) and t (supersink). Orient all edges of G from V_1 to V_2. Add edges from s to all sources of G, and from all sinks of G to t. All edges in the extended network have capacity 1.
2. Compute a maximum flow in the newly constructed flow network with source s and sink t.

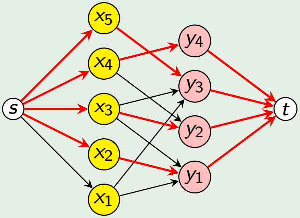
Example



Example

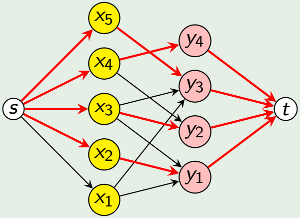


Example



Maximum matching $C = \{(x_2, y_1), (x_3, y_2), (x_4, y_4), (x_5, y_3)\}$

Example



Maximum matching $C = \{(x_2, y_1), (x_3, y_2), (x_4, y_4), (x_5, y_3)\}$

Theorem

Let G be the flow network constructed for a bipartite graph $B = (V_1 \cup V_2, E)$, and f a maximum flow in G computed with Ford-Fulkerson logarithm. Then the set of edges (u, v) of f with $u \in V_1$, $v \in V_2$ and $f(u, v) = 1$ is a maximum matching of B.

Problem

$G = (V, E)$: flow network in which every edge (u, v) has a capacity $c(u, v)$ and a unit cost $k(u, v) \geq 0$.
A maximum flow with minimum cost in G is a maximum flow f in G such that the sum

$$\sum_{(u,v)\in E} f(u, v) \cdot k(u, v)$$

is minimum.

Solution: Adjustment of Edmonds-Karp algorithm

- Attach costs to all edges of the residual networks of a flow f:
  - edge (u, v) has cost $k(u, v)$ if $c(u, v) > f(u, v)$ in the original flow network
  - edge (u, v) has cost $-k(u, v)$ if $f(u, v) < 0$ in the original flow network
- Instead of shortest simple path from source s to sink t, this algorithm finds a path p from s to t with minimum cost in the residual network.
  - p can be found with Bellman-Ford algorithm.
- Next, the flow is incremented along path p with the maximum possible value (=minimum of the differences between capacity and flow, for every arc of p).

References

Chapter 27 from

- T. H. Cormen, C. E. Leiserson, R. L. Rivest. *Introduction to Algorithms*. MIT Press, 2000.