Mathematical Foundations for Computer Science

probability and optimization

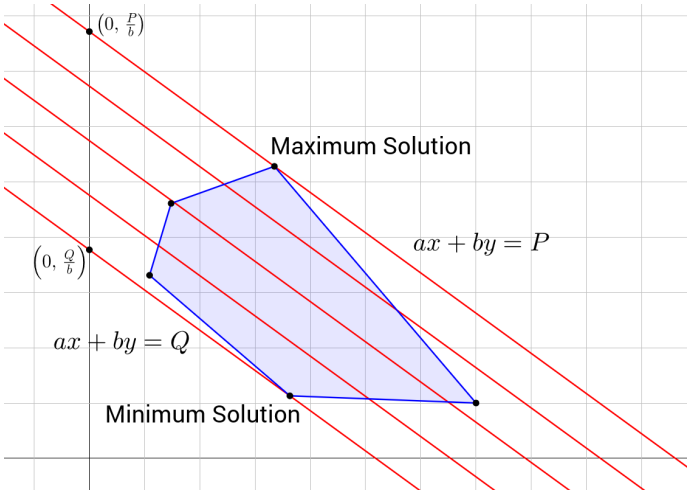
Chapter 5: Combinatorial Optimization

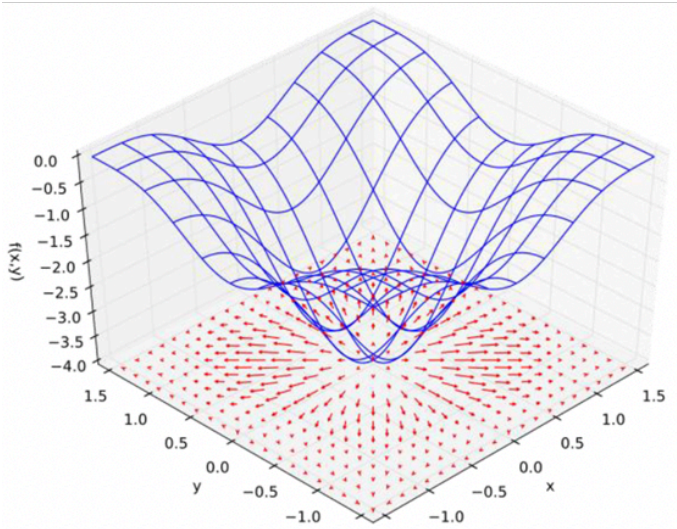
Spring 2021

Instructor: Xiaodong Gu

Recall: Convex Optimization

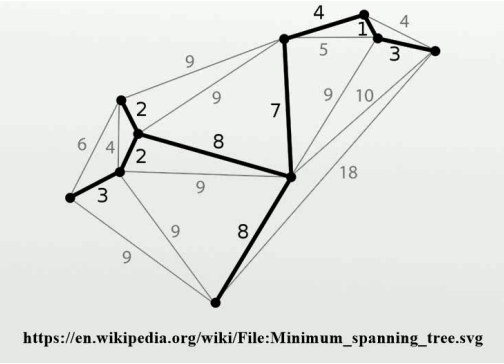
Find an optimal solution in a continuous and convex function, within a continuous and convex set.



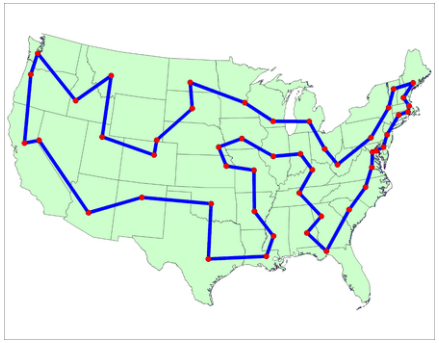


Combinatorial Problems

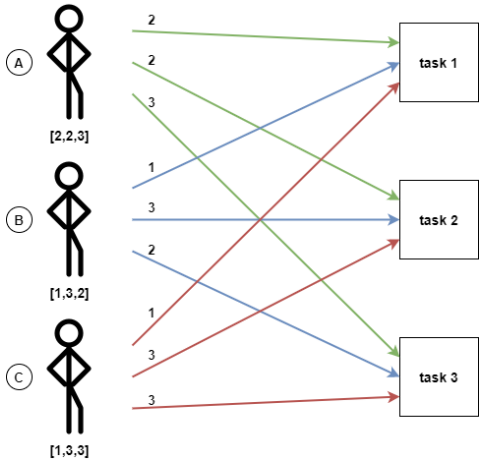
But there are optimization problems that cannot be characterized by continuous functions



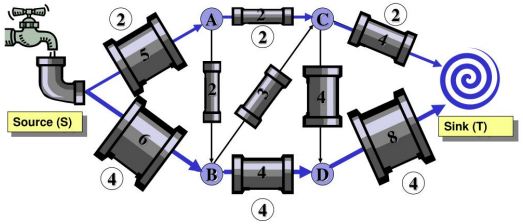
minimum spanning tree



minimum length tour that visits all cities



task assignments



maximize flow by arranging tubes

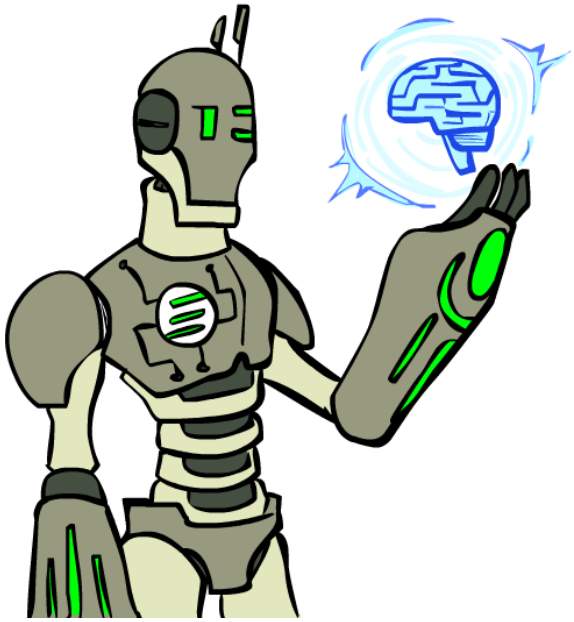
Combinatorial Optimization

- Combinatorial problems involve finding a grouping, ordering, or assignment of a discrete, finite set of objects that satisfies given conditions.
- Candidate solutions are combinations of solution components that may be encountered during a solutions attempt but need not satisfy all given conditions.
- Solutions are candidate solutions that satisfy all given conditions.

Overview of Chapter 5

Introduction to Combinatorial Optimization

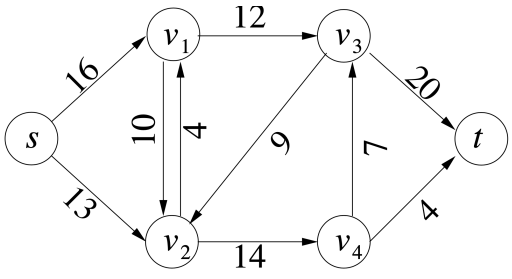
Maximum Flow and Minimum Cut, Bipartite Matching



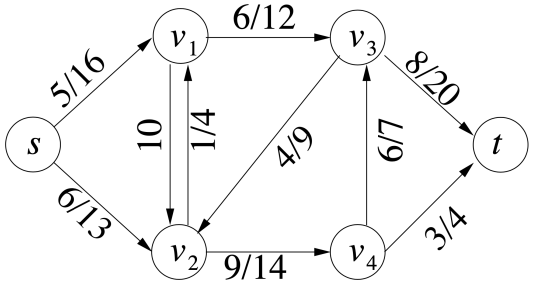
Maximum Flow

Maximum Flow

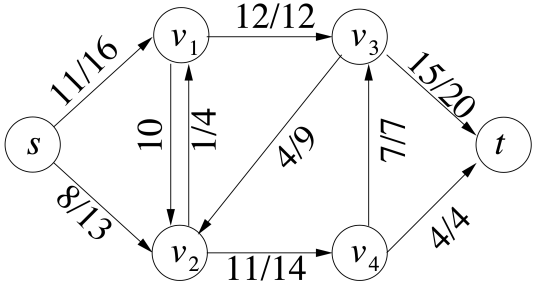
A flow network is a directed graph G = (V, E) with source s∈V and a sink t∈V. Every edge (u, v) ∈ E has a capacity c (u, v) ≥ 0.
Max-flow: find the maximum amount (flow) that can be shipped from s to t.



A flow network with capacities



A flow with value 11



A max-flow: value is 19

* flow is assumed to be discrete here. For example, goods, data packages,…

Problem Formulation

Assume that for every $v \in V$, there is a path from s to v and from v to t.

A FLOW is a function $f : V \times V \rightarrow R$ satisfying:

- capacity constraint: $\forall u, v \in V, f(u, v) \leq c(u, v)$.
- skew symmetry: $\forall u, v \in V, f(u, v) = -f(v, u)$.
- flow conservation: $\forall u \in V \setminus \{s, t\}, \ \Sigma_{v \in V} f(u, v) = 0$.

The VALUE of flow v is $|f| = \Sigma_{v \in V} f(s, v)$.

MAXIMUM-FLOW PROBLEM:

Given G, c, s, t, find f that maximizes $|f|$.

MAXIMUM-FLOW $\in$ NP-complete

Flows across Sets

Let $X, Y \subseteq V$. We define $f(X,Y) = \Sigma_{x\in X\, y\in Y} f(x,y)$.

The flow-conservation constraint then just says

$$\forall u \in V \setminus \{s,t\},\ f(u,V)=0.$$

Lemma: (proof omitted)

$\forall X \subseteq V$, $f(X,X) = 0$. $\forall X,Y \subseteq V$, $f(X, Y)= -f(Y, X)$.

$\forall X,Y, Z \subseteq V$ with $X\cap Y =\emptyset$ $f(X\cup Y, Z) = f(X, Z)+f(Y, Z)$ and $f(Z, X\cup Y) = f(Z, X) + f(Z, Y)$

- Flow *f* was defined as amount that leaves source s.
- We now see that this is the same as amount that enters sink t..

$\|f\| = f(s, V)$	definition
$= f(V, V)-f(V-s, V)$	lemma
$= -f(V-s, V)$	lemma
$= f(V, V-s)$	lemma
$= f(V, t)+f(V, V-s-t)$	lemma
$= f(V, t)$	flow conservation

A Key Idea of Combinatorial Optimization

In every optimization problem we have to deal with the question:

> How can we prove that our solution is optimal (maximal/minimal)?

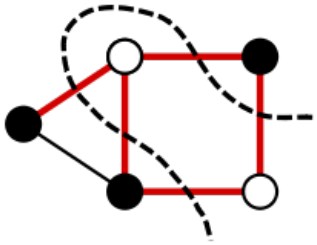
A common technique (for max problems) is to find a good upper-bound on the cost of an optimal solution and then show that our solution satisfies that bound.

For example, what is the upper bound of the maximal flow problem?

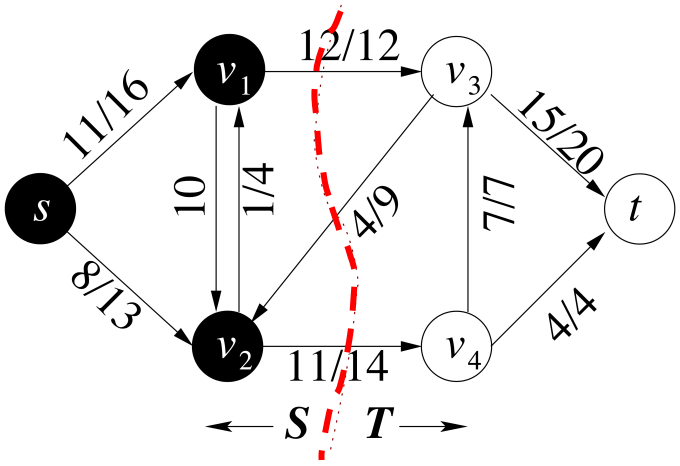
Max Flow vs. Min Cut

A CUT S, T of G is a partition of the vertices:

$V = S \cup T, S \cap T = \emptyset, s \in S$, and $t \in T$.



- The flow across the cut is $f(S, T)$.
- The capacity of a cut is $C(S, T) = \Sigma_{x \in S, y \in T} c(x, y)$.
- Note that for any cut, $f(S,T) \le C(S,T)$.



Cut (S,T): S = {s,v_1,v_2}, T = {v_3,v_4, *t*}.

The flow value is |f|=11+8=19, flow across the cut f(S,T) =12-4+11=19, and C(S,T) = 12+14 = 26.

Note that $|f| \le C(S,T)$.

Max Flow vs. Min Cut

Lemma:

If S, T is any cut, f is any flow then $|f| \leq C(S,T)$.

Proof:

$$
\begin{aligned}
|f| &= f(s, V) \\
&= f(s,V)+f(S-s,V) \\
&= f(S,V) \\
&= f(S,V)-f(S,S) \\
&= f(S,V-S) \\
&= f(S,T) \\
&\leq c(S,T)
\end{aligned}
$$

We will now develop the Ford-Fulkerson method for finding max-flows. When FF terminates it provides a flow f and a cut S,T such that |f| = C(S,T), so f is maximal.

The Ford-Fulkerson Algorithm

- Is iterative.
- Starts with flow $f = 0$, $(\forall u, v, f(u, v)=0)$
- At each step
  - constructs a residual network G_f of f indicating how much capacity "remains" to be used.
  - finds an augmenting path s-t path p in G_f along which flow can be pushed.
  - pushes f' units of flow along p. Creates new flow $f = f + f'$.
- Stops when there is no s-t path in current G_f.
- Let S be the set of nodes reachable from s in G_f and $T = V \setminus S$.

At conclusion of F.F. algorithm, $|f| = C(S, T)$, so f is optimal.

Residual Networks

Given flow f, the residual network G_f consists of the edges along which we can (still) push more flow. The amount that can (still) be pushed across (u, v) is called the residual capacity $c_f(u, v)$.

$$c_f(u, v) = c(u, v) - f(u, v).$$

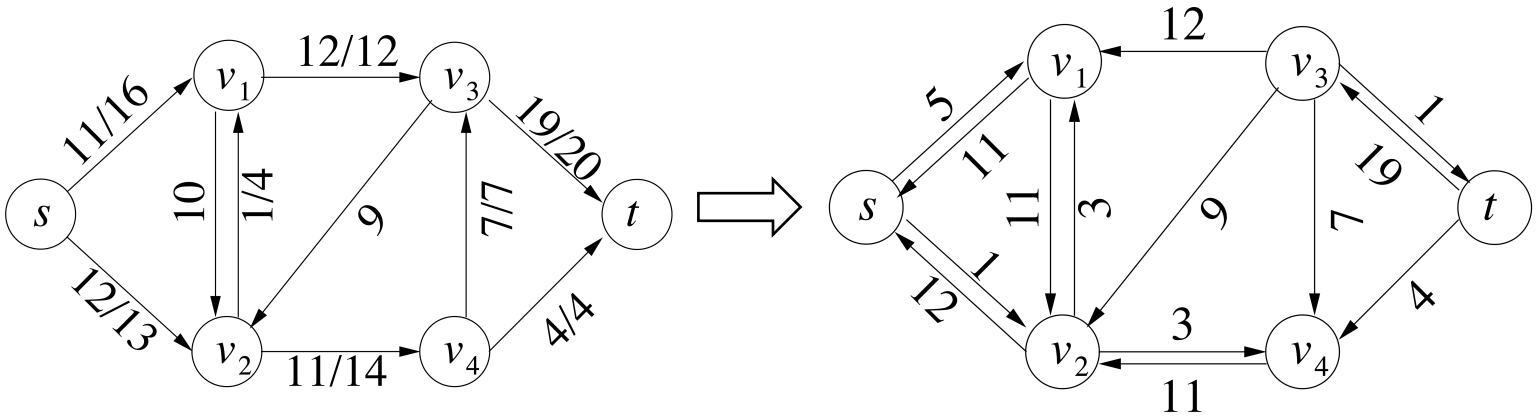
- If there is flow from u to v then $f(u,v) > 0$ and $c_f(u, v)$ is the remaining capacity on (u, v).
- If there is flow from v to u then $f(u, v) < 0$ and $c_f(u, v) = c(u,v) + f(v, u)$ is the capacity of (u, v) plus amount of existing flow that can be pushed backwards from u to v.

- The Residual Network G_f is $G_f = (V, E_f)$ where

$$E_f = \{(u, v) \in V \times V : c_f(u, v) > 0\}$$

Residual Networks

Example



A flow

Its residual network

Augmenting Path

Lemma:

Let f be a flow in G = (V, E) and G_f its residual network. Let f' be a flow in G_f. Define $f+f'$ as $(f+f')(u,v) = f(u,v) + f'(u,v)$.
Then $f+f'$ is a flow in G with value $|f + f'| = |f| + |f'|$.

Augmenting path p is a simple s-t path in G_f. The residual capacity of a.p. p is $c_f(p) = \min\{c_f(u, v) : (u, v) \text{ on } p\}$.

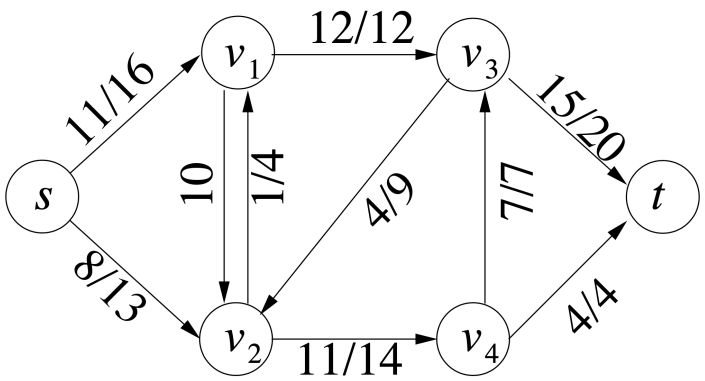
Let p be an augmenting path in G_f and define

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

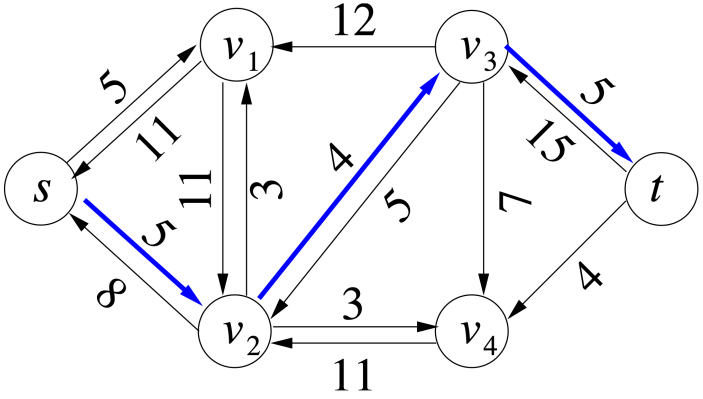
Lemma:

If f is a flow and p is an augmenting path in G_f then: f_p is a flow in G_f with $|f_p| = c_f(p) > 0$. $f' = f + f_p$ is a flow in G with $|f'| = |f| + |f_p| > |f|$.

Example

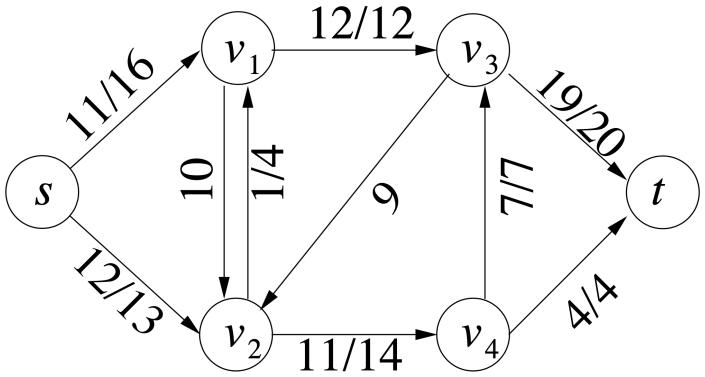


An initial flow f.

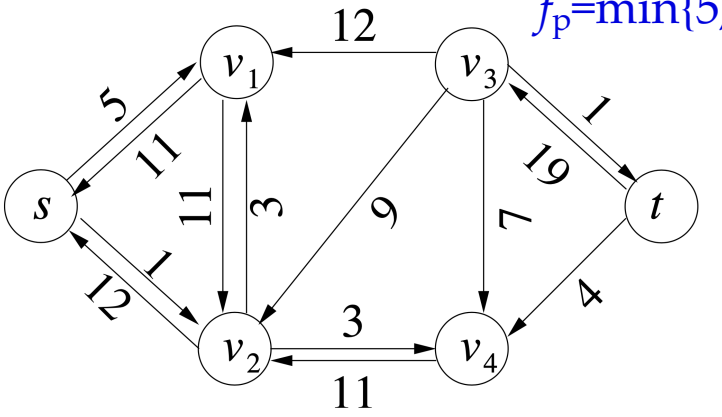


Its residual network G_f and an aug. path p.

f_p=min{5,4,5}=4



The flow $f + f_p$



Residual Network of $f + f_p$.

Optimality of Max Flow

Theorem: (Max-Flow Min-Cut Theorem)

Let *f* be a flow. Then the following three 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 (S,T) cut.

Proof:

- (1)⇒(2): If G_f contained an augmenting path p then $|f + f_p| > |f|$ so f could not be maximal.
- (2)⇒(3): Let S ={u∈V: ∃path from s to u in G_f}. T = V−S.

  Then $f(S,T) = f(S, V) - f(S, S) = f(S,V) = f(s, V) + f(S-s, V) = f(s, V) = |f|$. Now note that $\forall u \in S, v \in T, f(u, v) = c(u, v)$ since otherwise $c_f(u, v) > 0$ and $v \in S$ (contradicts to $v \in T$). Thus $C(S,T) = f(S,T) = |f|$.
- (3)⇒(1): We previously saw that every flow f' must satisfy $|f'| \le C(S,T)$ so if $|f| = C(S,T)$, f must be optimal.

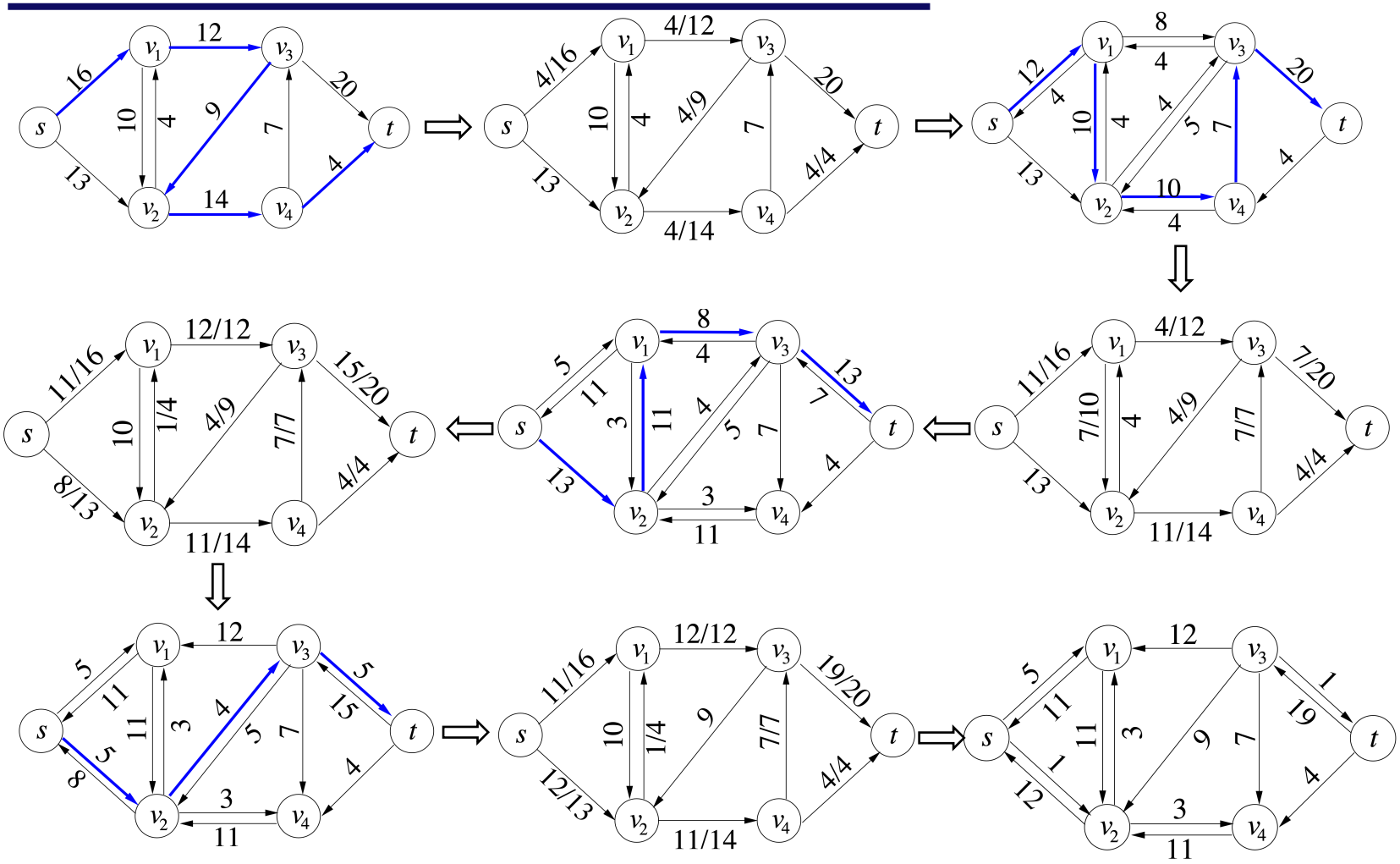
The Ford-Fulkerson Algorithm

- Starts with flow f ≡ 0, ($\forall u,v$,f(u,v)=0)
- Construct residual network G_f :
  If G_f contains no augmenting path:
  stop (f is optimal by MFMC theorem).

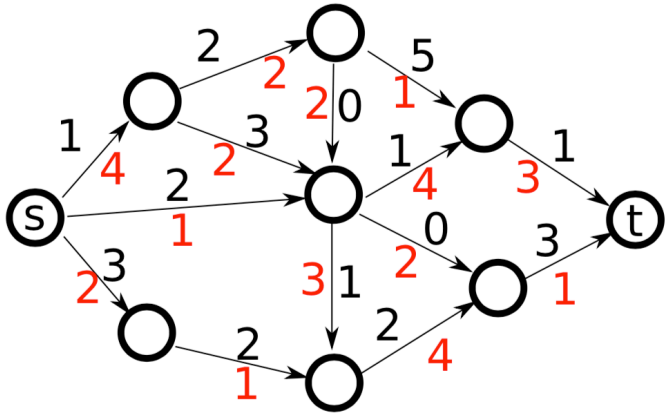
  Otherwise:

  1. Find an augmenting path (s–t path) p in G_f
  2. Let f_p be the flow in G_f that pushes $c_f(p)$ units of flow along p.
  3. Let $f = f + f_p$ be new flow in G.

Example


s
t
v_1
v_2
v_3
v_4
16
13
12
14
10
4
9
7
20
4
4/16
4/12
4/9
4/4
4/14
8
12
5
10
11/16
12/12
1/4
15/20
8/13
11/14
7/7
7/10
7/20
3
11
13
19/20
12/13
1
19

The ILP of Max Flow and Min Cut



Let p denote any *s-t* path, let P={p} denote all *s-t* paths:

Max Flow

$$\begin{array}{lll} \max & \Sigma_{p\in P} x_p & \\ \text{s.t.} & \Sigma_{p\in P,\, e\in p} x_p \le c(e) & \forall e\in E \\ & x_p \ge 0 & \forall p\in P \\ & x_p \in Z & \forall p\in P \end{array}$$

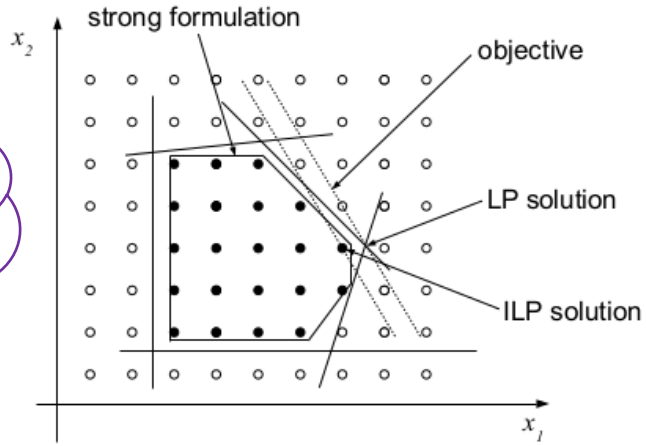
Min Cut

$$\begin{array}{lll} \min & \Sigma_{e\in E} c(e) y_e & \\ \text{s.t.} & \Sigma_{e\in p} y_e \ge 1 & \forall p\in P \\ & y_e \ge 0, & \forall e\in E \\ & y_e \in Z, & \forall e\in E \end{array}$$

Integer Linear Programming (ILP)

- Integer programming is an optimization problem in which some of the variables are restricted to be integers. In many settings the term refers to integer linear programming (ILP), in which the objective function and the constraints are linear.
- ILP is NP-complete
- So we usually find an approximate solution using LP-relaxation

Another idea to solve combinatorial optimization



ILP and LP- relaxation

Lemma*

The optimal value of ILP is equal to that of the associated LP-relaxation if the linear constraint matrix A is totally unimodular (i.e., every square submatrix of A has a determinant of 0, 1, or -1).

Primal-Dual

Max Flow

$$\begin{aligned} \max \quad & \Sigma_{p\in P} x_p \\ \text{s.t.} \quad & \Sigma_{p\in P,\, e\in p} x_p \le c(e) \quad \forall e\in E \\ & x_p \ge 0 \quad \forall p\in P \end{aligned}$$

Min Cut

$$\begin{aligned} \min \quad & \Sigma_{e\in E} c(e) y_e \\ \text{s.t.} \quad & \Sigma_{e\in p} y_e \ge 1 \quad \forall p\in P \\ & y_e \ge 0, \quad \forall e\in E \end{aligned}$$

Weak Duality: $\Sigma_{p\in P} x_p \le \Sigma_{e\in E} c(e) y_e$

Strong Duality: $\Sigma_{p\in P} x_p = \Sigma_{e\in E} c(e) y_e$ (max flow = min cut)

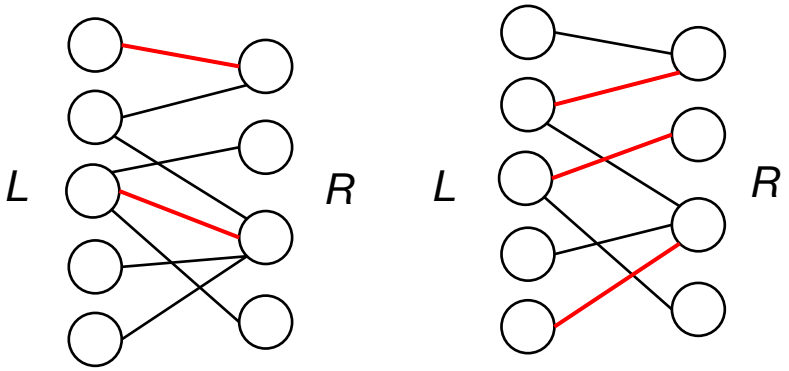
The Ford-Fulkerson method finds the condition of strong duality

Application: Max Bipartite Matching

Application: Max Bipartite Matching

- A graph $G = (V, E)$ is bipartite if there exists partition $V = L \cup R$ with $L \cap R = \emptyset$ and $E \subseteq L \times R$.
- A matching is a subset $M \subseteq E$ such that $\forall v \in V$ at most one edge in M is incident upon v. The size of a matching $|M|$ is the number of edges in M.

Example: A bipartite graph with 2 matchings

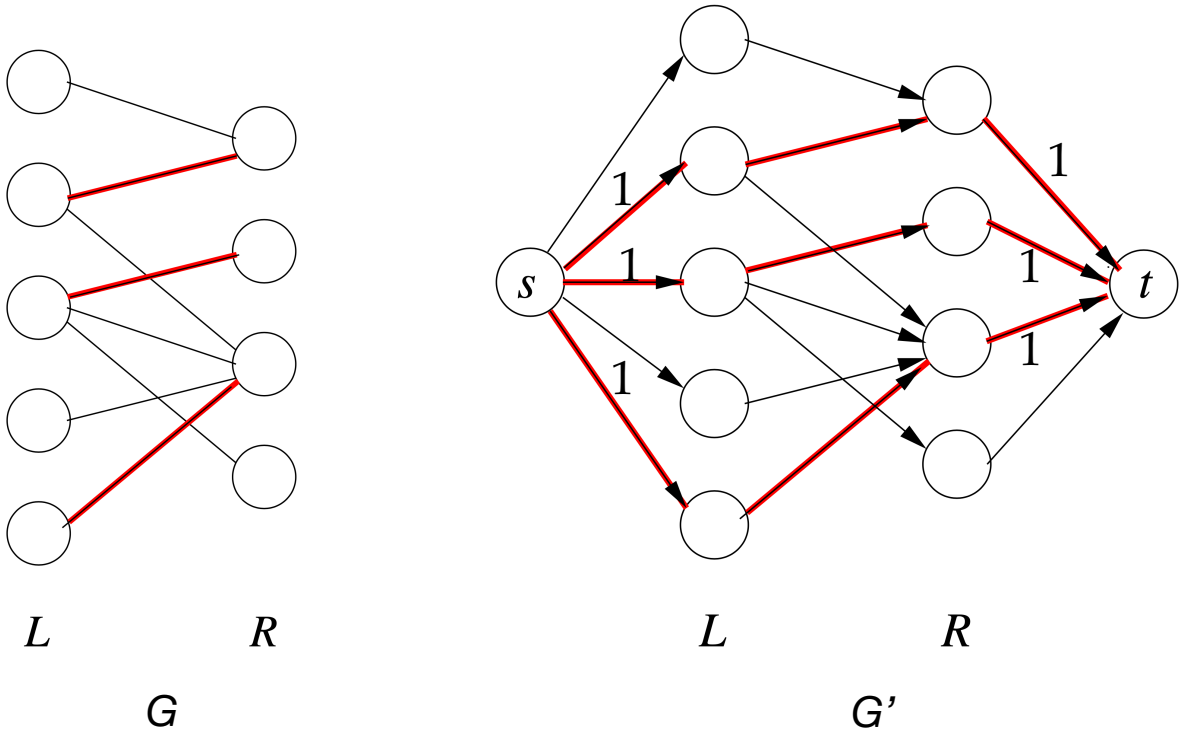


- A maximum matching is matching M such that every other matching M' satisfies $|M'| \leq M$.

Problem: given bipartite graph G, find a maximum matching.

Idea: Max Bipartite Matching as Max-Flow

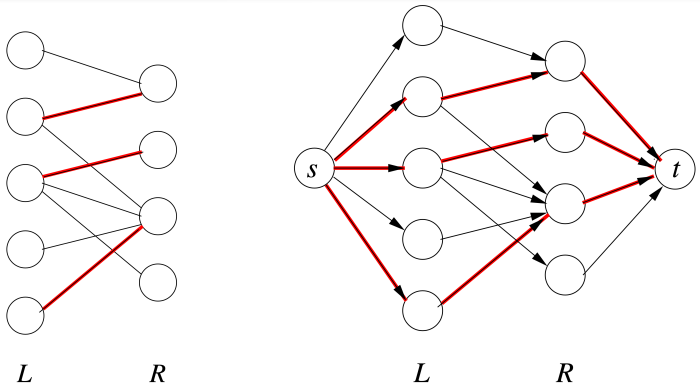
Construct a flow network $G' = (V', E')$ where $V' = V \cup \{s, t\}$ and $E' = \{(s, u): u \in L\} \cup \{(u, v) : u \in L, v \in R \text{ and } (u, v) \in E\} \cup \{(v, t): t \in R\}$. We also assign $\forall (u, v) \in E'$, $c(u, v) = 1$.



Max-Flow Solves Max-Matching

Lemma

If f is an integer valued flow in G′ then there is a matching M of G with $|f| = |M|$.
Similarly, if M is a matching of G then there is an integer valued flow f with $|f| = |M|$.



- This almost tells us that max-flow solves max matching.
- The difficulty is that it's possible that the max-flow might not have integer value (it is possible that $|f|$ might be an integer but some $f(u, v)$ might not be integers).

Integral Max Flow

Theorem

Let G′ = (V′, E′) be a flow network in which $c(u, v)$ is integral. Then the max-flow f found by the F.F. method has the property that $\forall u, v, f(u, v)$ is integer valued.

- The proof is by induction on the steps in the F.F. method. At each step the current flow f is integer so the residual capacities are all integer. This implies that the a.p. found has $c_f(p)$ integral so the new flow $f + f'$ created is also integral.
- The theorem guarantees that if G′ is the flow network corresponding to a bipartite matching problem then max flow value $|f|$ is the value of a maximum matching.
- The flow found by the F.F. algorithm can be modified to yield the max matching.