# COS 226 Lecture 20: Network Flow

Advanced graph problems
- NETWORK FLOW
- MATCHING

OPERATIONS RESEARCH

Classical problems (1940s)
- Graph algorithm technology
- PQ and data structure design crucial
- Optimal solutions still not known

Examples of REDUCTION
- BIPARTITE MATCHING
  - no harder than network flow
  - (actually, much less general)
- LINEAR PROGRAMMING
  - no easier than network flow
  - (actually, much more general)

Network flow, matching, linear programming
- are each fields of study in their own right

# Network flow

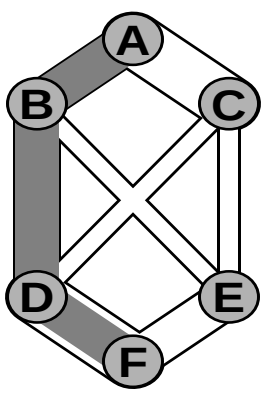
NETWORK: weighted graph

Consider network as abstraction for material
FLOWING through the edges

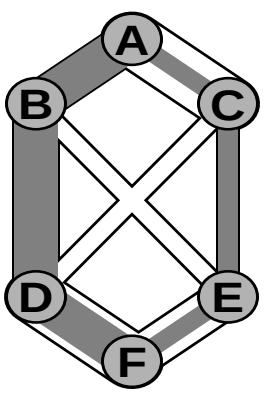
Interpret weights as CAPACITIES
nodes as FLOW CONTROL SWITCHES
Ex: oil flowing in pipes
Ex: commodities flowing on roads and rails
Ex: bits flowing in Internet

Identify
SOURCE: node where all material originates
SINK: node where all material goes

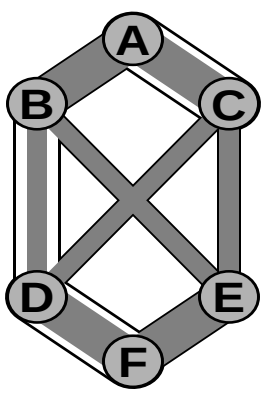
MAX FLOW PROBLEM: assign flows to edges that
maximize total flow through the network




dumb




naive




best

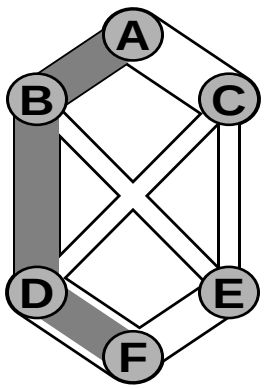
# Increasing flow in a network

Assume directed graph: all links one-way
(downhill in examples)

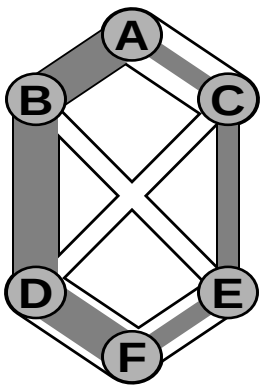
AUGMENTING PATH:
path from source to sink with unused capacity on every edge

Easy case
all edges on path carry positive flow
(up to capacity)

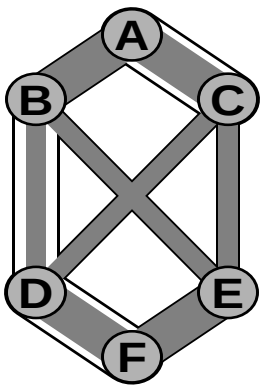
More complicated case
negative flow in BACKWARD edges
(up to amount of flow)




ABDF




ACEF




ACDBEF

# Ford-Fulkerson algorithm

Start with 0 flow everywhere

Increase the flow along any augmenting path
until no augmenting paths are left

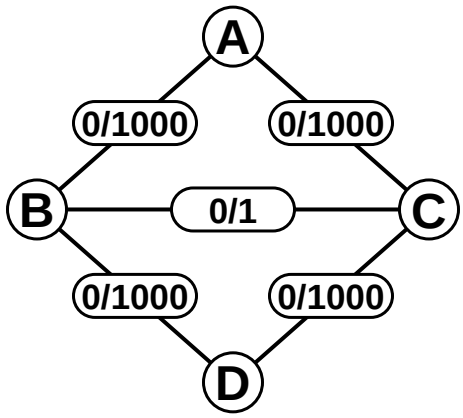
Problem 0:
Does this lead to the maximum flow?

Problem 1:
Not really an algorithm, since
many details are unspecified
* How do we find an augmenting path?
* Which augmenting path is best?

Problem 2:
Cost can be proportional to max capacity

Ex:

ABCD
ACBD
ABCD
ACBD
ABCD
...



## Max flow/Min cut theorem

CUT: set of edges separating source from sink

THM: Finding the maximum flow is equivalent to finding the minimum cut

Proof: Omitted

THM: Ford-Fulkerson method gives maximum flow

Proof sketch:

If there is no augmenting path,
identify the first full forward
or empty backward edge on every path
That set of edges defines a min cut

Applies to an entire family of algorithms

No matter how the augmenting paths
are chosen, the flow is maximal
when no more can be found

Algorithm design goals

* find paths quickly
* use as few iterations as possible

## Edmonds-Karp algorithms

Idea 1:

use BFS to find augmenting path

Idea 2:

find path that increases the flow the most

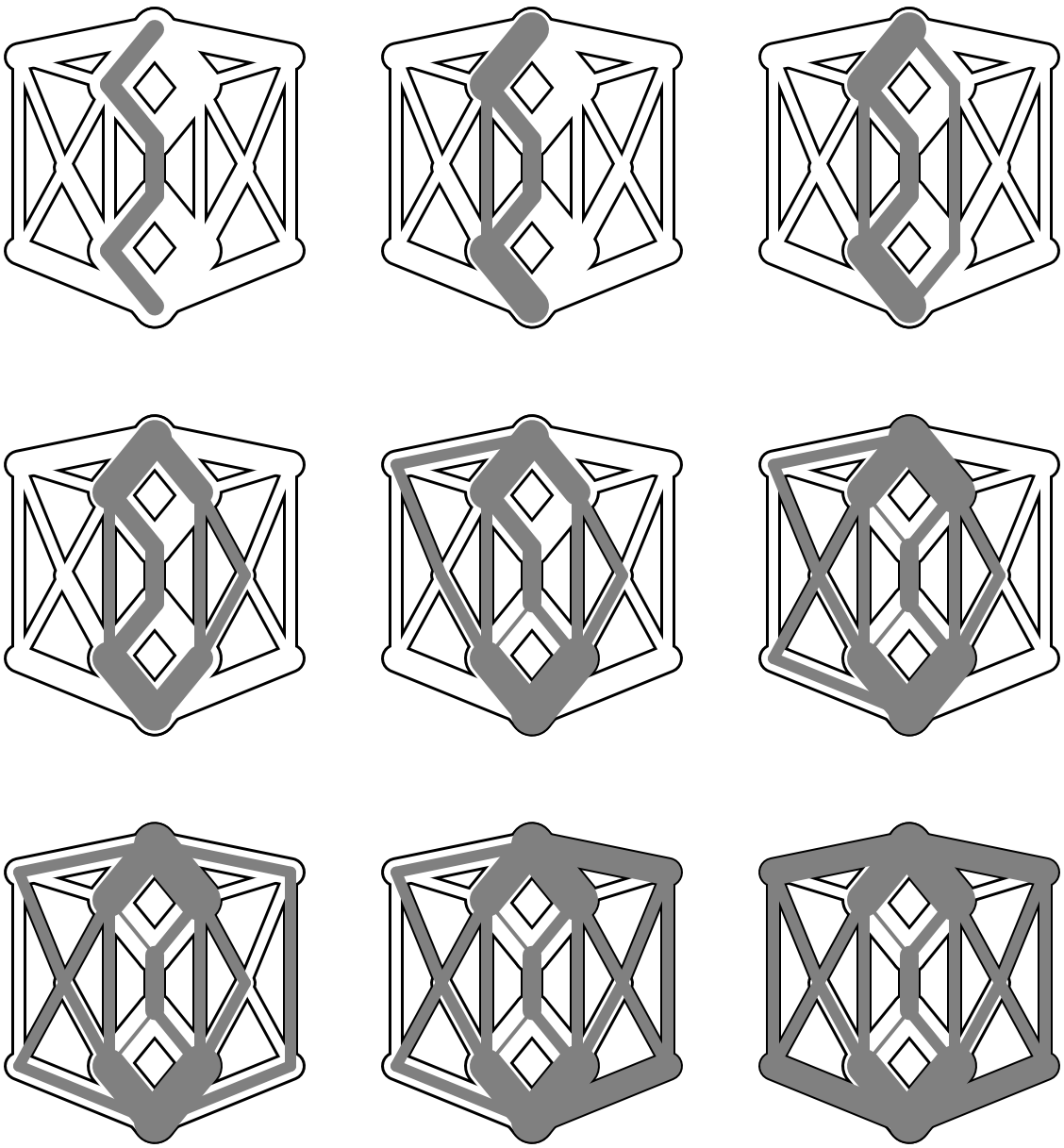
BOTH easy to implement with PQ graph search (!)

Change basic graph search routine to

* use positive weights for forward path,
  negative weights for backward path
* ignore full forward, empty backward edges
* take source, sink as args, return 0 if no path

```
for (;;)
  {
    if (!matrixpfs(1,V)) break;
    y = V; x = dad[V];
    while (x != 0)
      {
        flow[x][y] = flow[x][y]+val[V];
        flow[y][x] = -flow[x][y];
        y = x; x = dad[y];
      }
  }
```

# Network flow example



## Analysis of network flow algorithms

Edmonds-Karp worst-case running time

- BFS: V E^2
  - (VE iterations, each cost E)
- best path: C(M, F) E log V
  - depends on cut size and flow amount
  - (M/(M-1))^C = F
- Ex: M = 1000, F = 1000: C = 6900
- Ex: M = 10, F = 1000: C = 72

New algorithms

- * use better priority queue implementations
- * remember facts about search for last path
  - to find new path
- * find multiple paths on one search

Running times

- 1970 V^2 E
- 1977 V^2 E^(1/2)
- 1978 V^3
- 1978 V^(5/3) E^(2/3)
- 1980 V E log V
- 1986 V E log(V^2/E)

Objection!

- real graphs not "average" or "worst case"
- simple (but not dumb) algorithms may be
  - preferred in practice

BUT simple optimal algorithm could still exist

# Matching

MATCHING

set of edges with no vertex included twice

MAXIMUM MATCHING

no matching contains more edges

WEIGHTED MATCHING

weighted graph

no matching has greater sum of edge weights

BIPARTITE GRAPH

two types of nodes

all edges connect nodes of differing type

BIPARTITE MATCHING

maximum matching in bipartite graph

Ex:

graph: E5 A2 A1 C1 B4 C3 D3 B2 A4 D5 E3 B1

matching: A1 B4 C3 D5 E6 F2

ASSIGNMENT PROBLEM

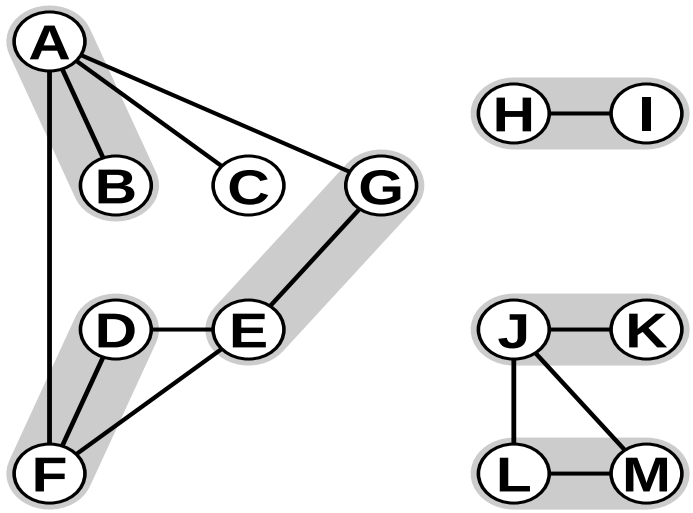
weighted bipartite matching

Matching is difficult to solve in general

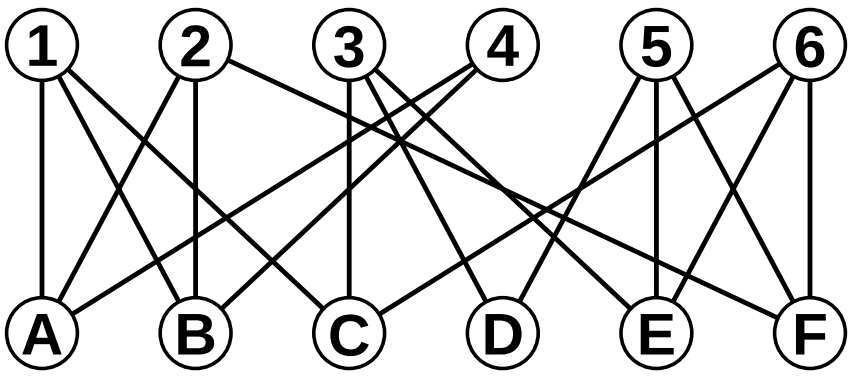
Related to network flow

# Matching examples

Matching




Bipartite graph



# Reducing A to B

SOLUTION: efficient algorithm that can
give the answer for any problem instance

Ex: Given any network, Edmonds-Karp algorithms
solve network flow problem for that network

REDUCTION: a way to solve problems by converting
them to other problems

Ex: Bipartite matching reduces to network flow
Convert instance of bipartite matching to
instance of network flow
Solve network flow instance
Convert network flow solution to bipartite
matching solution

Ex: Network flow reduces to linear programming
(stay tuned)

Reduction is not necessarily EFFICIENT
Ex: Everything computable reduces to
Universal Turing Machine

BUT reduction, in practice,

* is better than brute force
* is better than investing years of research
to understand details of a problem
* gives insight into difficulty
(partially order problems by difficulty)

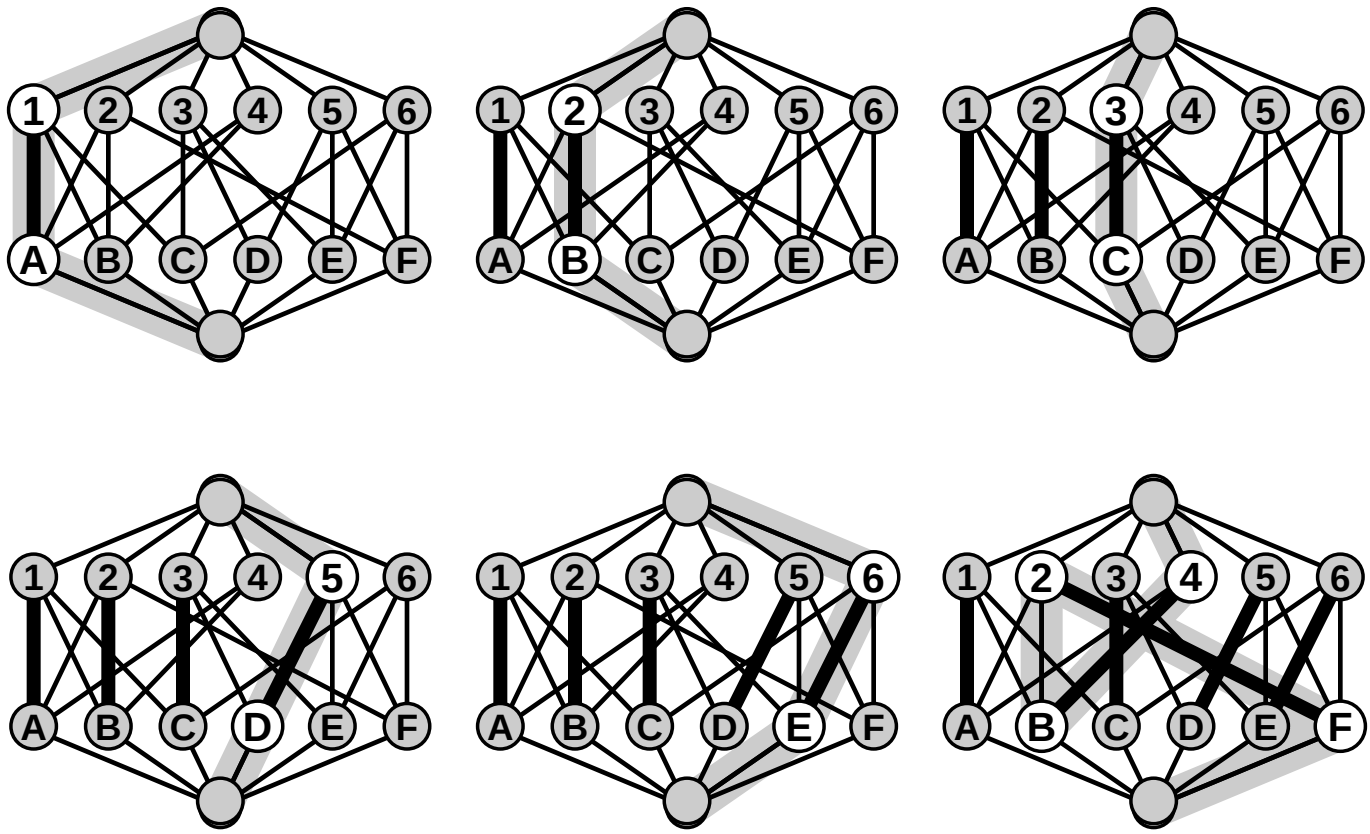
## Reducing bipartite matching to network flow

Create two new nodes

SOURCE with edges to all nodes of first type

SINK with edges to all nodes of second type

Make all capacities 1




Network flow easier for unit capacities

NOTE: Reduction doesn't work for weighted case

(assignment problem)

# Linear programming

Maximize OBJECTIVE FUNCTION
subject to CONSTRAINTS

Ex:

maximize x+y
subject to the constraints

$$-x + y \leq 5$$
$$x + 4y \leq 45$$
$$2x + y \leq 27$$
$$3x - 4y \leq 24$$
$$x, y \geq 0$$

Vast number of applications

Early application: DIET problem
min cost diet with adequate nutrients

"Programming": formulate the equations
(reduce given problem to linear programming)

## Reducing network flow to linear programming

Express capacities as inequalities

flow at nodes as equalities

Ex:

maximize AB + AD

subject to the constraints

AB <= 6

CD <= 3

AC <= 8

CE <= 3

BD <= 6

DF <= 8

BE <= 3

EF <= 6

BD + BE = AB

CD + CE = AC

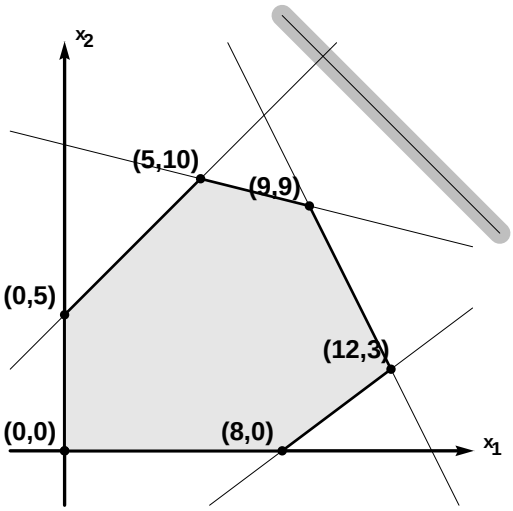
BD + CD = DF

BE + CE = EF

all >= 0

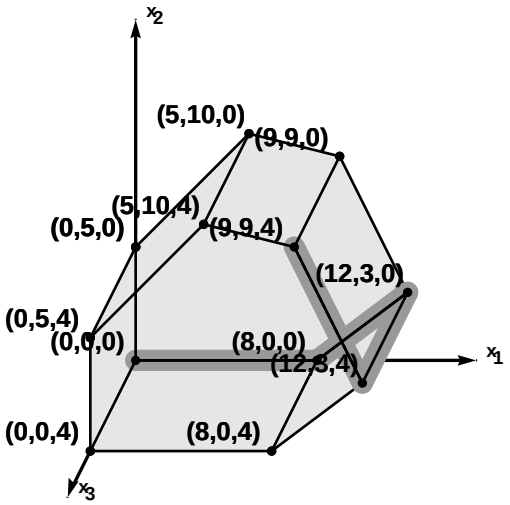
# Geometric interpretation

Ex: (two variables)




objective function maximized at vertex

Ex: (three variables)




Higher dimensions

multifacted convex polytope: SIMPLEX

objective function maximized at vertex

## Approaches to linear programming

Simplex method

- classical approach
- like Gaussian elimination
- pivot steps correspond to moving on simplex vertices
- exponential in worst case
- fast in practice

Duality

- is point inside simplex?

Ellipsoid methods

- new algorithmic approach
- geometric divide-and-conquer
- polynomial time (!)

Challenge:

- make new methods faster than simplex for practical problems

## Perspective

Nearing the deep waters of NP-completeness

Ex: INTEGER programming NP complete

Ex: TRIpartite matching NP-complete

1960s Operations research

1970s Analysis of algorithms

1980s Geometric algorithms

All relevant to wide variety of
important practical problems

Consider ALL approaches for new problems

Reduction

- may help determine complexity
- may provide usable solution
  - for practical instances

Specialized algorithms

- may provide elegant optimal solution
  - for all instances

General methods

- may solve specific practical instance
  - quickly (even if can't be proven fast)