COS 226 Lecture 22: Mincost Flow

MAXFLOW: assign flows to edges that

- equalize inflow and outflow at every vertex
- maximize total flow through the network

MINCOST MAXFLOW: find the BEST maxflow

Mincost maxflow is important for two primary reasons

it is a GENERAL PROBLEM-SOLVING MODEL

- solves (through reduction) numerous practical problems

it is TRACTABLE and PRACTICAL

- we know fast algorithms that solve mincost flow problems
- basic data structures play a critical role

One step closer to a single ADT for combinatorial problems

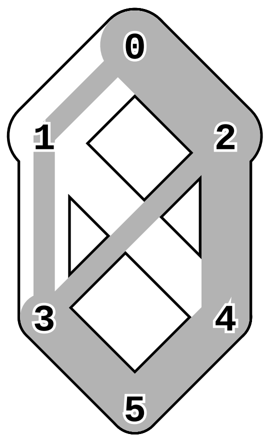
Mincost flow

Add COST to each edge in a flow network

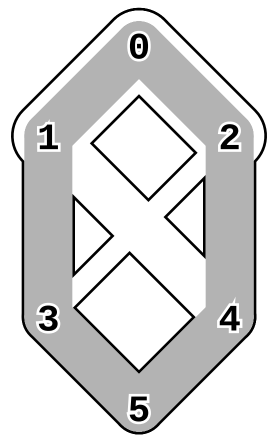
FLOW COST: sum of cost*flow over all edges

Maxflows have different costs

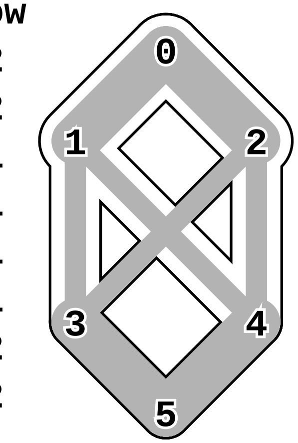
	cap	cost	flow	flow	flow
0-1	3	3	1	2	2
0-2	3	1	3	2	2
1-3	2	1	1	2	1
1-4	2	1	0	0	1
2-3	1	4	1	0	1
2-4	2	2	2	2	1
3-5	2	2	2	2	2
4-5	2	1	2	2	2



cost: 21



cost: 20



cost: 22

MINCOST FLOW: find a minimal-cost maxflow

Distribution problem

SUPPLY vertices (produce goods)

DEMAND vertices (consume goods)

DISTRIBUTION points (transfer goods)

Feasible flow problem

- Can we make supply to meet demand?

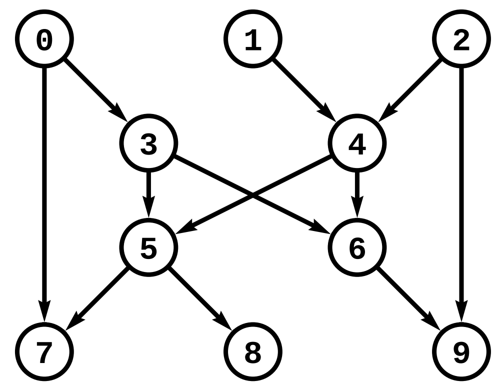
Distribution problem

- Add costs, find the lowest-cost way

Ex: Walmart

Ex: McDonald's

supply		channels	
0:	3	0-3:	2
1:	4	0-7:	1
2:	6	1-4:	5
distribution		2-4:	3
3		2-9:	1
4		3-5:	3
5		3-6:	4
6		4-5:	2
demand		4-6:	1
7:	7	5-7:	6
8:	3	5-8:	3
9:	4	6-9:	4



THM: Feasible flow reduces to maxflow

THM: Distribution reduces to mincost maxflow

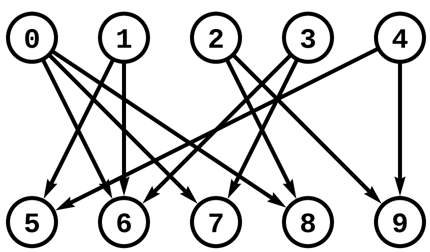
Proof: Add source to provide supply, sink to take demand

Transportation problem

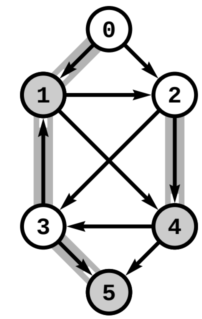
No distribution points

- feasibility: is there a way?
- transportation: find best way

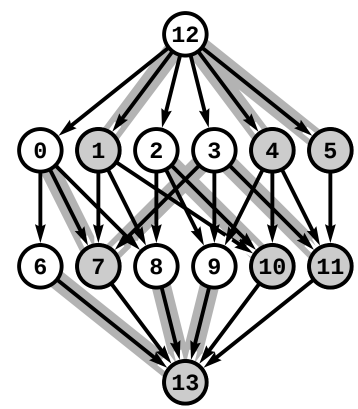
supply		channels	
0:	3	0-6:	2
1:	4	0-7:	1
2:	6	0-8:	5
3:	3	1-6:	3
4:	2	1-5:	1
demand		2-8:	3
5:	6	2-9:	4
6:	6	3-6:	2
7:	7	3-7:	1
8:	3	4-9:	6
9:	4	4-5:	3



0-1	2
0-2	3
1-2	3
1-4	2
2-3	2
2-4	1
3-1	3
3-5	2
4-3	3
4-5	3



0-6	25	12-0	5	6-13	5
1-7	25	12-1	5	7-13	5
2-8	25	12-2	3	8-13	3
3-9	25	12-3	5	9-13	5
4-10	25	12-4	6	10-13	6
5-11	25	12-5	0	11-13	0
0-7	2				
0-8	3				
1-8	3				
1-10	2				
2-9	2				
2-10	1				
3-7	3				
3-11	2				
4-9	3				
4-11	3				



Seems easier, but that is not the case (!)

THM: Maxflow reduces to maxflow for acyclic networks

THM: Transportation reduces to mincost maxflow

Mincost flow reductions

SHORTEST PATHS

MAXFLOW

DISTRIBUTION and TRANSPORTATION

ASSIGNMENT

Minimal weight matching in weighted bipartite graph

MAIL CARRIER

Find a cyclic path that includes each edge AT LEAST once

SCHEDULING (example)

Given a sport's league schedule, which teams are eliminated?

POINT MATCHING

Given two sets of N points, find minimal-distance pairing

ALL of these problems reduce to mincost flow

Cycle canceling

RESIDUAL NETWORK

for each edge in original network

- flow f in edge u-v with capacity c and cost x

define TWO edges in residual network

- FORWARD edge: capacity c-f and cost x in edge u-v
- BACKWARD edge: capacity f and cost -x in edge v-u

THM: A maxflow is mincost iff
there are NO negative-cost cycles in its residual network

GENERIC method for solving mincost flow problems:

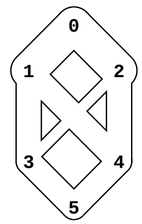
start with ANY maxflow
REPEAT until no negative cycles are left

- increase the flow along ANY negative cycle

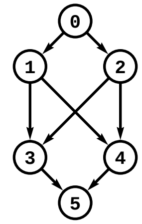
Implementation: use Bellman-Ford to find negative cycles

Cycle canceling example

	cap	cost	flow
0-1	3	3	0
0-2	3	1	0
1-3	2	1	0
1-4	2	1	0
2-3	1	4	0
2-4	2	2	0
3-5	2	2	0
4-5	2	1	0



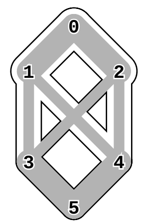
0-1	3
0-2	3
1-3	2
1-4	2
2-3	1
2-4	2
3-5	2
4-5	2



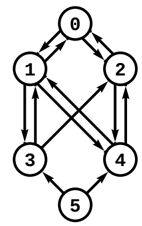
initial maxflow

	cap	cost	flow
0-1	3	3	2
0-2	3	1	2
1-3	2	1	1
1-4	2	1	1
2-3	1	4	1
2-4	2	2	1
3-5	2	2	2
4-5	2	1	2

total cost: 22



0-1	1	1-0	2
0-2	1	2-0	2
1-3	1	3-1	1
1-4	1	4-1	1
		3-2	1
2-4	1	4-2	1
		5-3	2
		5-4	2

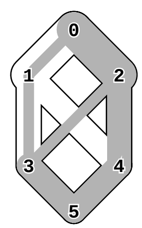


negative cycles: 4-1-0-2-4
3-2-0-1-3
3-2-4-1-3

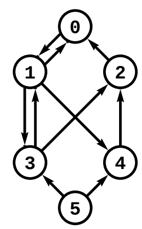
augment +1 on 4-1-0-2-4 (cost -1)

	cap	cost	flow
0-1	3	3	1
0-2	3	1	3
1-3	2	1	1
1-4	2	1	0
2-3	1	4	1
2-4	2	2	2
3-5	2	2	2
4-5	2	1	2

total cost: 21



0-1	2	1-0	1
		2-0	3
1-3	1	3-1	1
1-4	2		
		3-2	1
		4-2	2
		5-3	2
		5-4	2

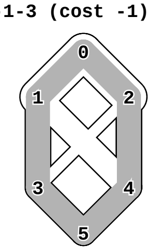


negative cycle: 3-2-0-1-3

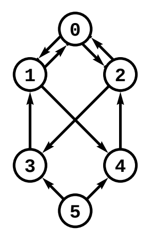
augment +1 on 3-2-0-1-3 (cost -1)

	cap	cost	flow
0-1	3	3	2
0-2	3	1	2
1-3	2	1	2
1-4	2	1	0
2-3	1	4	0
2-4	2	2	2
3-5	2	2	2
4-5	2	1	2

total cost: 20



0-1	1	1-0	2
0-2	1	2-0	2
		3-1	2
1-4	2		
2-3	1		
2-4	1	4-2	2
		5-3	2
		5-4	2



Cycle canceling implementation

```
void addflow(link u, int d)
  { u->flow += d; u->dup->flow -=d; }
int GRAPHmincost(Graph G, int s, int t)
  { int d, x, w; link u, st[maxV];
    GRAPHmaxflow(G, s, t);
    while ((x = GRAPHnegcycle(G, st)) != -1)
      {
        u = st[x]; d = Q;
        for (w=u->dup->v; w != x; w=u->dup->v)
        { u = st[w]; d = ( Q > d ? d : Q ); }
        u = st[x]; addflow(u, d);
        for (w=u->dup->v; w != x; w=u->dup->v)
        { u = st[w]; addflow(u, d); }
      }
    return GRAPHcost(G);
  }
```

Cycle canceling analysis

No need to compute initial maxflow

- use dummy edge from sink to source that carries maxflow

THM: Generic cycle canceling alg takes O(VE^2CM) time

Proof:

- each edge has at most capacity C and cost M
- total cost could be ECM
- each augment reduces cost by at least 1
- Bellman-Ford takes O(VE) time

There exist O(VE^2log^2 V) cycle-canceling implementations

- mincost maxflow is therefore TRACTABLE

EXTREMELY pessimistic UPPER bounds

- not useful for predicting performance in practice
- algs that achieve such bounds would be useless
- algs are typically fast on practical problems

Network simplex algorithm

An implementation of the cycle-canceling algorithm

Identify negative cycles quickly by

- maintaining a tree data structure
- reweighting costs at vertices

Edge classification

- EMPTY
- FULL
- PARTIAL

FEASIBLE SPANNING TREE

- Any spanning tree that contains all the partial edges

VERTEX POTENTIALS

- a set of vertex weights (vertex-indexed array phi)

Network simplex concepts (continued)

REDUCED COST (reweighted edge cost)

- c*(u, v) = c(u, v) - (phi(u) - phi(v))

VALID vertex potentials for a spanning tree

- all tree edges have reduced cost 0

ELIGIBLE EDGE

- nontree edge that creates negative cycle with tree edges

THM: A nontree edge is eligible iff it is either

- a full edge with positive reduced cost, or
- an empty edge with negative reduced cost

Proof:

- cycle cost equals cycle reduced cost
- edge cost is negative of cycle reduced cost

  (since reduced costs of tree edges are all zero)

THEREFORE, it is easy to identify eligible edges

Network simplex algorithm

still a generic algorithm for the mincost flow problem

start with ANY feasible spanning tree

REPEAT until no eligible edges are left

- ensure that vertex potentials are valid
- add to the tree an eligible edge
- increase the flow along the negative cycle formed
- remove from the tree an edge that is filled or emptied

Problem: could have zero flow on cycle

THM: IF the algorithm terminates, it computes a maxflow

Implementation challenges

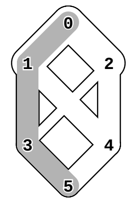
- cope with zero-flow cycles
- strategy to choose eligible edges
- data structure to represent tree

Network simplex example

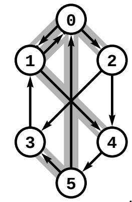
add 3-5, augment +2 on 0-1-3-5-0, del 1-3

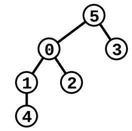
	cap	cost	flow
0-1	3	3	2
0-2	3	1	0
1-3	2	1	2
1-4	2	1	0
2-3	1	4	0
2-4	2	2	0
3-5	2	2	2
4-5	2	1	0
0-5	6	9	2

flow cost: 30



	cost*	
0-1		1-0
0-2		2-0
1-3	-3	3-1
1-4		4-1
2-3	-2*	3-2
2-4	-1*	4-2
3-5		5-3
4-5	-4*	5-4
5-0		5-0



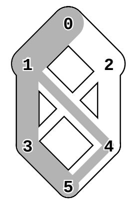


	0	1	2	3	4	5
st	5	0	0	5	1	5
pt	0	-3	-1	-7	-4	-9

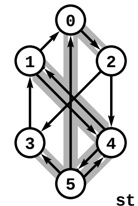
add 4-5, augment +1 on 0-1-4-5-0, del 0-1

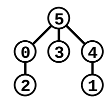
	cap	cost	flow
0-1	3	3	3
0-2	3	1	0
1-3	2	1	2
1-4	2	1	1
2-3	1	4	0
2-4	2	2	0
3-5	2	2	2
4-5	2	1	1
0-5	6	9	1

flow cost: 26



	cost*	
0-1	-4	1-0
0-2		2-0
1-3	1*	3-1
1-4		4-1
2-3	-2*	3-2
2-4	-5*	4-2
3-5		5-3
4-5		5-4
		5-0



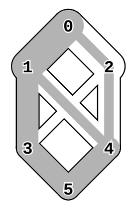


	0	1	2	3	4	5
st	5	4	0	5	5	5
pt	0	-7	-1	-7	-8	-9

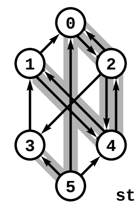
add 2-4, augment +1 on 0-2-4-5-0, del 4-5

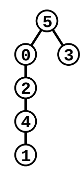
	cap	cost	flow
0-1	3	3	3
0-2	3	1	1
1-3	2	1	2
1-4	2	1	1
2-3	1	4	0
2-4	2	2	1
3-5	2	2	2
4-5	2	1	2
0-5	6	9	0

flow cost: 21



	cost*	
0-1	1*	1-0
2-0		2-0
1-3	-4	3-1
1-4		4-1
2-3	-2*	3-2
2-4		4-2
3-5		5-3
4-5	-5	5-4
		5-0



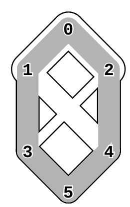


	0	1	2	3	4	5
st	5	4	0	5	2	5
pt	3	1	0	0	-3	-4

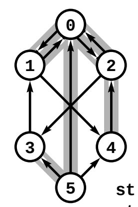
add 1-0, augment +1 on 1-0-2-4-1, del 4-1

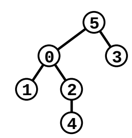
	cap	cost	flow
0-1	3	3	2
0-2	3	1	2
1-3	2	1	2
1-4	2	1	0
2-3	1	4	0
2-4	2	2	2
3-5	2	2	2
4-5	2	1	2
0-5	6	9	0

flow cost: 20



	cost*	
0-1		1-0
0-2		2-0
1-3	-3	3-1
1-4	1	4-1
2-3	-2*	3-2
2-4		4-2
3-5		5-3
4-5	-5	5-4
		5-0



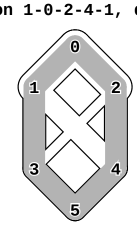


	0	1	2	3	4	5
st	5	0	0	5	2	5
pt	0	-3	-1	-7	-3	-9

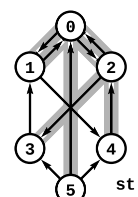
add 2-3, augment 0 on 1-0-2-4-1, del 3-5

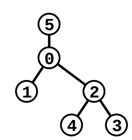
	cap	cost	flow
0-1	3	3	2
0-2	3	1	2
1-3	2	1	2
1-4	2	1	0
2-3	1	4	0
2-4	2	2	2
3-5	2	2	2
4-5	2	1	2
0-5	6	9	0

flow cost: 20



	cost*	
0-1		1-0
0-2		2-0
1-3	-1	3-1
1-4	1	4-1
2-3		3-2
2-4		4-2
3-5	-2	5-3
4-5	-5	5-4
		5-0





	0	1	2	3	4	5
st	5	0	0	2	2	5
pt	0	-3	-1	-5	-3	-9

Feasible spanning tree data structure

Operations to support

- compute valid vertex potentials
- find cycle created by nontree edge
- replace tree edge by nontree edge

use PARENT-LINK representation!

to compute vertex potentials

- start with root at potential 0
- for each vertex

    follow parent links to vertex with known potential

    (recursively) set each vertex potential on path

    to make reduced edge costs 0

to follow cycle created by nontree edge u-v

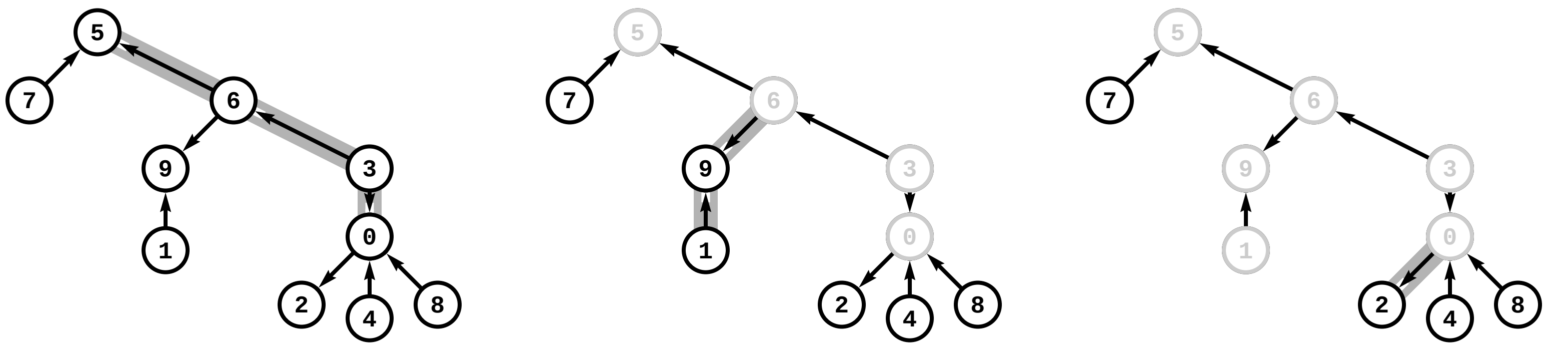
- follow parent links from each to their LCA

to delete nontree edge that fills or empties

- REVERSE the parent links from u or v

Computing vertex potentials (example)

	cost
3-0	3
1-9	3
0-2	2
3-6	2
4-0	3
7-5	2
8-0	2
6-9	2
6-5	2

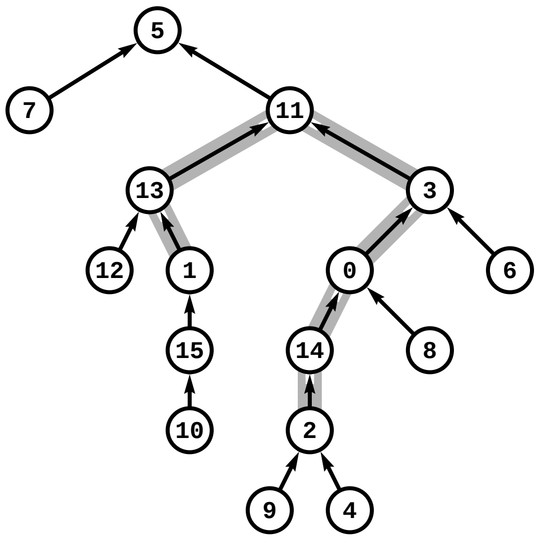


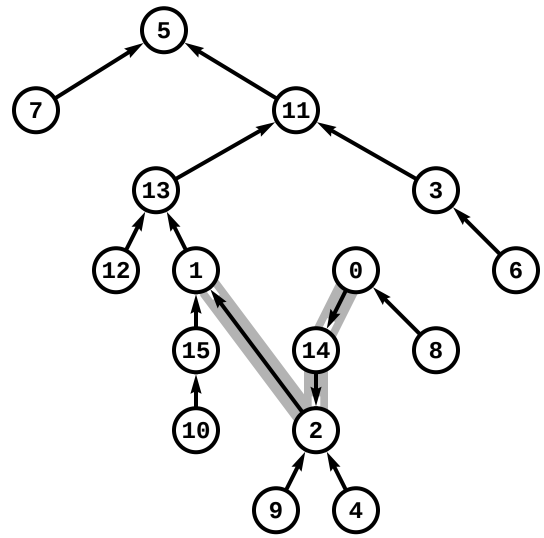
	0	1	2	3	4	5	6	7	8	9
pt	1			4		0	2			

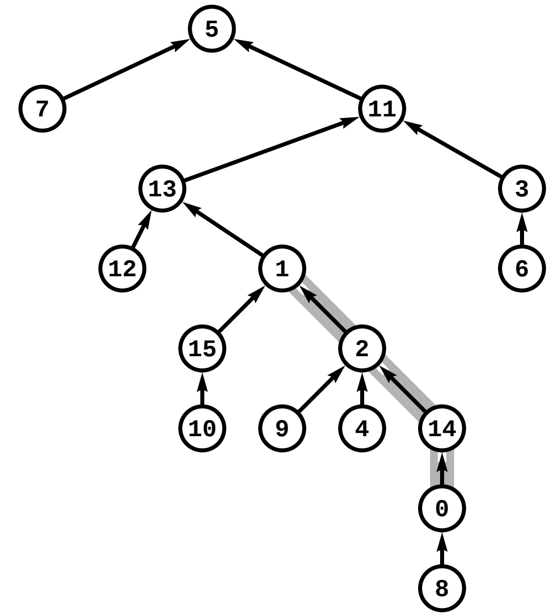
	0	1	2	3	4	5	6	7	8	9
pt	1	3		4		0	2			0

	0	1	2	3	4	5	6	7	8	9
pt	1	3	-1	4		0	2			0

Spanning tree update example







	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
st	3	13	14	11	2	5	3	5	0	2	15	5	13	11	0	1

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
st	14	13	1	11	2	5	3	5	0	2	15	5	13	11	2	1

Network simplex basic implementation

```
#define R(u)  u->cost - phi[u->v]+phi[u->dup->v]
int GRAPHmincost(Graph G, int s, int t)
{ int v; link u, x, st[maxV];
  GRAPHinsertE(G, EDGE(t, s, M, 0, C));
  initialize(G, s, t, st);
  for (valid = 1; valid++; )
  {
    for (v = 0; v < G->V; v++)
      phi[v] = phiR(st, v);
    for (v = 0, x = G->adj[v]; v < G->V; v++)
      for (u = G->adj[v]; u!=NULL; u = u->next)
         if (R(u) < R(x)) x = u;
    if (R(x) == 0) break;
    update(st, augment(st, x), x);
  }
  return GRAPHcost(G);
}
```

Network simplex variations

OBJECTIVES

- guarantee terminimation
- reduce number of iterations
- reduce cost per iteration

Eligible edge selection strategies

- random
- find next
- queue of eligible edges

Lazy vertex potential calculation

Tree representations

- triply-linked, threaded

Guided by practical performance, not worst-case bounds

- DATA STRUCTURES are the key to good performance

Different implementations for different reductions??

BOTTOM LINE

- accessible code for powerful problem-solving model