MAX FLOW APPLICATIONS

CS302, Spring 2013 David Kauchak

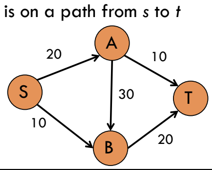
Admin

- CS lunch today
- Grading

Flow graph/networks

Flow network

- directed, weighted graph (V, E)
- positive edge weights indicating the "capacity" (generally, assume integers)
- contains a single source $s \in V$ with no incoming edges
- contains a single sink/target $t \in V$ with no outgoing edges
- every vertex is on a path from *s* to *t*

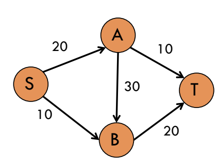


Flow constraints

in-flow = out-flow for every vertex (except s, t)

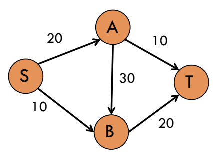
flow along an edge cannot exceed the edge capacity

flows are positive



Max flow problem

Given a flow network: *what is the maximum flow we can send from s to t that meet the flow constraints?*



Network flow properties

If one of these is true then all are true (i.e. each implies the the others):

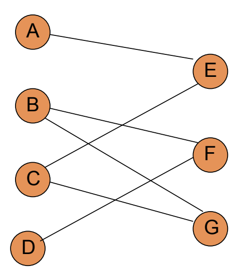
- f is a maximum flow
- G_f (residual graph) has no paths from s to t
- $|f|$ = minimum capacity cut

Ford-Fulkerson

Ford-Fulkerson(G, s, t)
  flow = 0 for all edges
  G_f = residualGraph(G)
  while a simple path exists from s to t in G_f
    send as much flow along the path as possible
    G_f = residualGraph(G)
  return flow

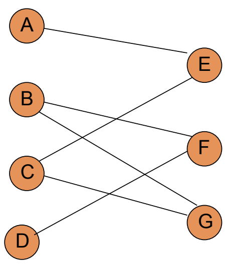
Application: bipartite graph matching

Bipartite graph – a graph where every vertex can be partitioned into two sets X and Y such that all edges connect a vertex $u \in X$ and a vertex $v \in Y$



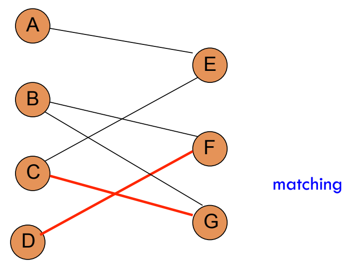
Application: bipartite graph matching

A *matching* M is a subset of edges such that each node occurs **at most once** in M



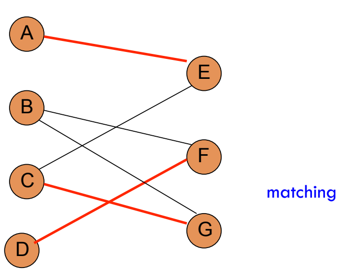
Application: bipartite graph matching

A *matching* M is a subset of edges such that each node occurs **at most once** in M



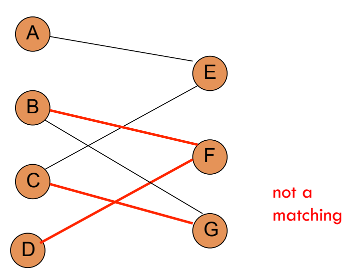
Application: bipartite graph matching

A *matching* M is a subset of edges such that each node occurs **at most once** in M



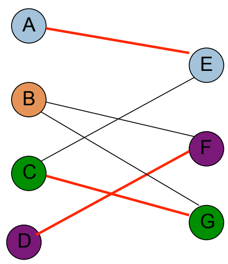
Application: bipartite graph matching

A *matching* M is a subset of edges such that each node occurs **at most once** in M



Application: bipartite graph matching

A *matching* can be thought of as pairing the vertices

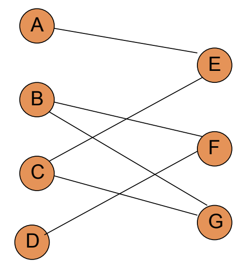


Application: bipartite graph matching

Bipartite matching problem: find the *largest* matching in a bipartite graph

Where might this problem come up?

- CS department has n courses and m faculty
- Every instructor can teach *some* of the courses
- What course should each person teach?
- Anytime we want to match n things with m, but not all things can match

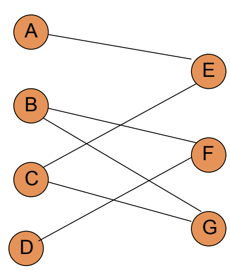


Application: bipartite graph matching

Bipartite matching problem: find the *largest* matching in a bipartite graph

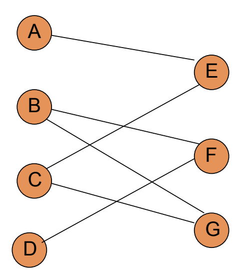
ideas?

- greedy?
- dynamic programming?



Application: bipartite graph matching

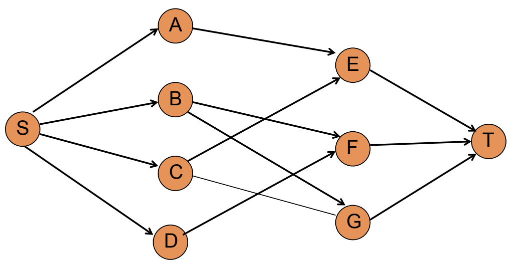
Setup as a flow problem:



Application: bipartite graph matching

Setup as a flow problem:

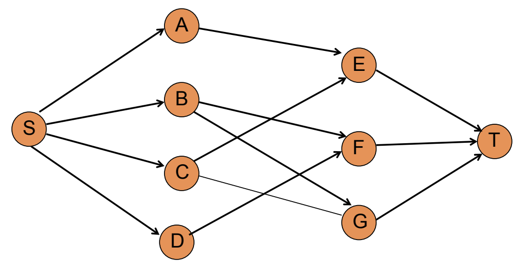
edge weights?



Application: bipartite graph matching

Setup as a flow problem:

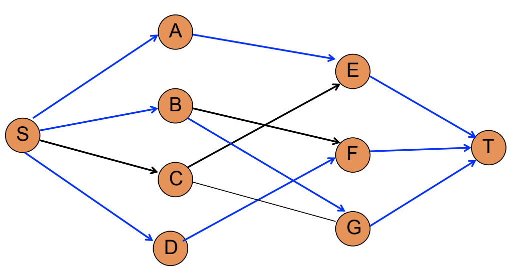
all edge weights are 1



Application: bipartite graph matching

Setup as a flow problem:

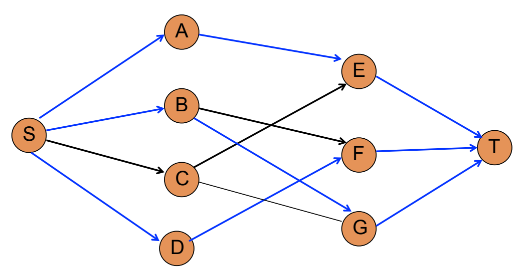
after we find the flow, how do we find the matching?



Application: bipartite graph matching

Setup as a flow problem:

match those nodes with flow between them



Application: bipartite graph matching

Is it correct?

Assume it's not

- there is a better matching
- because of how we setup the graph flow = # of matches
- therefore, the better matching would have a higher flow
- contradiction (max-flow algorithm finds maximal!)

Application: bipartite graph matching

Run-time?

Cost to build the flow?

- O(E)
  - each existing edge gets a capacity of 1
  - introduce V new edges (to and from s and t)
  - V is O(E) (for non-degenerate bipartite matching problems)

Max-flow calculation?

- Basic Ford-Fulkerson: O(max-flow * E)
- Edmunds-Karp: $O(V E^2)$
- Preflow-push: $O(V^3)$

Application: bipartite graph matching

Run-time?

Cost to build the flow?

- O(E)
  - each existing edge gets a capacity of 1
  - introduce V new edges (to and from s and t)
  - V is O(E) (for non-degenerate bipartite matching problems)

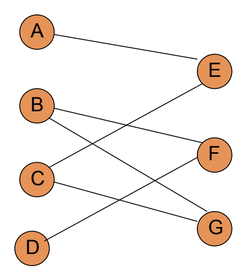
Max-flow calculation?

- Basic Ford-Fulkerson: O(max-flow * E)
  - max-flow = O(V)
  - O(V E)

Application: bipartite graph matching

Bipartite matching problem: find the *largest* matching in a bipartite graph

- CS department has n courses and m faculty
- Every instructor can teach *some* of the courses
- What course should each person teach?
- Each faculty can teach at most 3 courses a semester?



Change the s edge weights (representing faculty) to 3

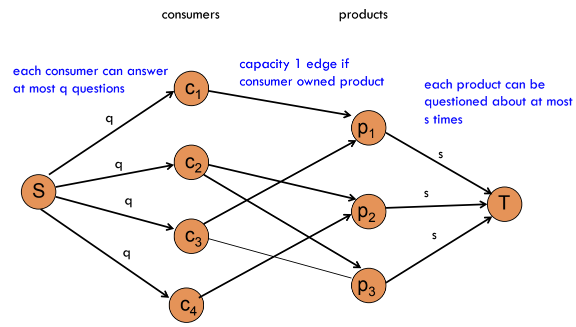
Survey Design

Design a survey with the following requirements:

- Design survey asking *n* consumers about *m* products
- Can only survey consumer about a product if they own it
- Question consumers about at most *q* products
- Each product should be surveyed at most *s* times
- Maximize the number of surveys/questions asked

How can we do this?

Survey Design



Survey design

Is it correct?

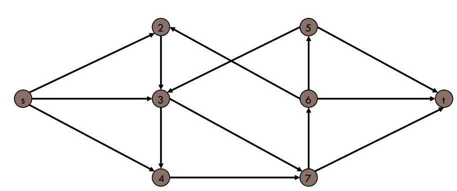
- Each of the comments above the flow graph match the problem constraints
- max-flow finds the maximum matching, given the problem constraints

What is the run-time?

- Basic Ford-Fulkerson: O(max-flow * E)
- Edmunds-Karp: $O(V E^2)$
- Preflow-push: $O(V^3)$

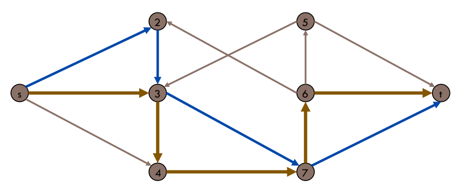
Edge Disjoint Paths

Two paths are edge-disjoint if they have no edge in common



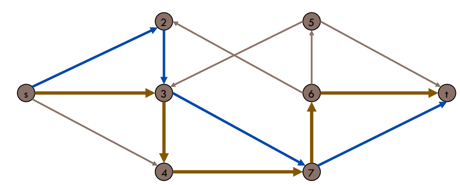
Edge Disjoint Paths

Two paths are edge-disjoint if they have no edge in common



Edge Disjoint Paths Problem

Given a directed graph G = (V, E) and two nodes s and t, find the max number of edge-disjoint paths from s to t



Why might this be useful?

Edge Disjoint Paths Problem

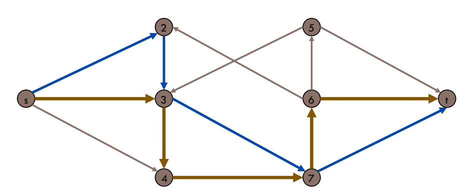
Given a directed graph G = (V, E) and two nodes s and t, find the max number of edge-disjoint paths from s to t

Why might this be useful?

- edges are unique resources (e.g. communications, transportation, etc.)
- how many *concurrent (non-conflicting)* paths do we have from s to t

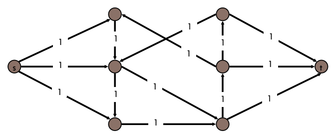
Edge Disjoint Paths

Algorithm ideas?



Edge Disjoint Paths

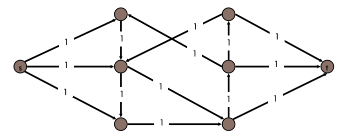
Max flow formulation: assign unit capacity to every edge



What does the max flow represent?
Why?

Edge Disjoint Paths

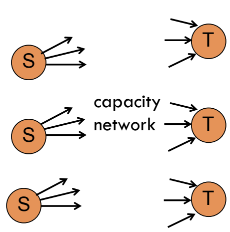
Max flow formulation: assign unit capacity to every edge



- max-flow = maximum number of disjoint paths
- correctness:
  - each edge can have at most flow = 1, so can only be traversed once
  - therefore, each unit out of s represents a separate path to t

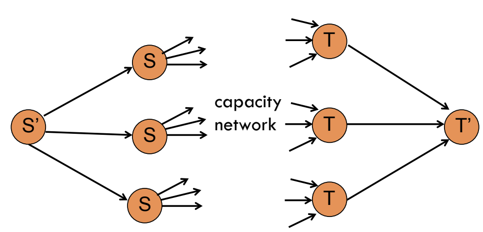
Max-flow variations

What if we have multiple sources and multiple sinks (e.g. the Russian train problem has multiple sinks)?



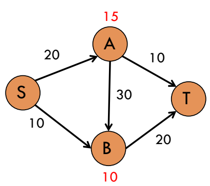
Max-flow variations

Create a new source and sink and connect up with infinite capacities…



Max-flow variations

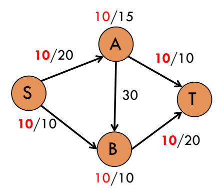
Vertex capacities: in addition to having edge capacities we can also restrict the amount of flow through each vertex



What is the max-flow now?

Max-flow variations

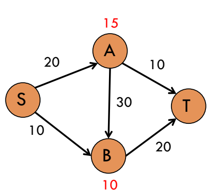
Vertex capacities: in addition to having edge capacities we can also restrict the amount of flow through each vertex



20 units

Max-flow variations

Vertex capacities: in addition to having edge capacities we can also restrict the amount of flow through each vertex

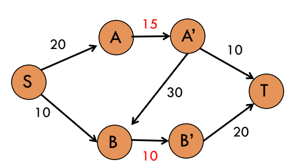


How can we solve this problem?

Max-flow variations

For each vertex v

- create a new node v'
- create an edge with the vertex capacity from v to v'
- move all outgoing edges from v to v'



Can you now prove it's correct?

Max-flow variations

Proof:

1. show that if a solution exists in the original graph, then a solution exists in the modified graph
2. show that if a solution exists in the modified graph, then a solution exists in the original graph

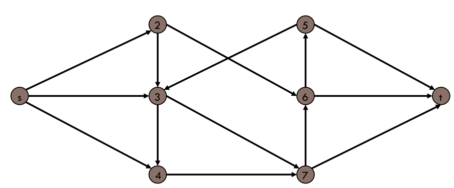
Max-flow variations

Proof:

- we know that the vertex constraints are satisfied
  - no incoming flow can exceed the vertex capacity since we have a single edge with that capacity from v to v'
- we can obtain the solution, by collapsing each v and v' back to the original v node
  - in-flow = out-flow since there is only a single edge from v to v'
  - because there is only a single edge from v to v' and all the in edges go in to v and out to v', they can be viewed as a single node in the original graph

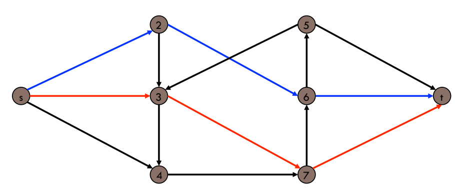
More problems: maximum independent path

Two paths are independent if they have no *vertices* in common



More problems: maximum independent path

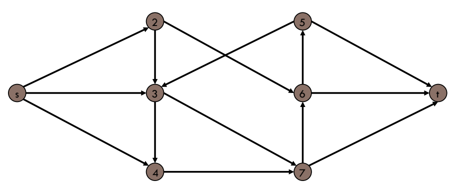
Two paths are independent if they have no *vertices* in common



More problems: maximum independent path

Find the maximum number of independent paths

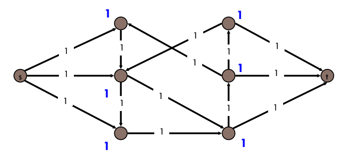
Ideas?



maximum independent path

Max flow formulation:

- assign unit capacity to every edge (though any value would work)
- assign unit capacity to every vertex



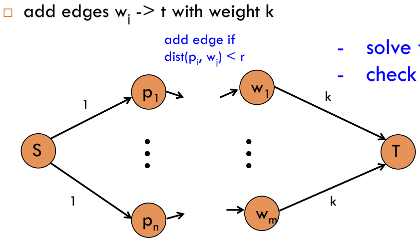
Same idea as the maximum edge-disjoint paths, but now we also constrain the vertices

More problems: wireless network

- The campus has hired you to setup the wireless network
- There are currently *m* wireless stations positioned at various (x,y) coordinates on campus
- The range of each of these stations is *r* (i.e. the signal goes at most distance *r*)
- Any particular wireless station can only host *k* people connected
- You've calculate the *n* most popular locations on campus and have their (x,y) coordinates
- Could the current network support n different people trying to connect at each of the *n* most popular locations (i.e. one person per location)?
- Prove correctness and state run-time

Another matching problem

- n people nodes and m station nodes
- if dist(p_i,w_j) < r then add an edge from pi to wj with weight 1 (where dist is euclidean distance)
- add edges s -> p_i with weight 1
- add edges w_j -> t with weight k



- solve for max-flow
- check if flow = m

Correctness

If there is flow from a person node to a wireless node then that person is attached to that wireless node

if dist(pi,wj) < r then add an edge from pi to wj with weigth 1 (where dist is euclidean distance)

- only people able to connect to node could have flow

add edges s -> pi with weight 1

- each person can only connect to one wireless node

add edges wj -> t with weight L

- at most L people can connect to a wireless node

If flow = m, then every person is connected to a node

Runtime

E = O(mn): every person is within range of every node

V = m + n + 2

max-flow = O(m), s has at most m out-flow

- O(max-flow * E) = $O(m^2n)$: Ford-Fulkerson
- $O(VE^2) = O((m+n)m^2n^2)$: Edmunds-Karp
- $O(V^3) = O((m+n)^3)$: preflow-push variant