CSC373: Algorithm Design, Analysis and Complexity
Winter/Spring 2020

Allan Borodin and Sara Rahmati

Week of February 3-7, 2020

Announcements and this weeks agenda

- We are posting Sara Rahmati's lecture slides. However, as Sara is using many slides from Kevin Wayne's web site, for copyright reasons we are password protecting Sara's slides. We will repeat the password in lecture.
- This week will be devoted to flow networks and an application to bipartite matching.
- Piazza has a form to help find teammates if you want to be in a team and have not teamed up yet.
- Accessability services is looking for someone to share notes taken during the lecture. Please see link on web page under the course news for February 2.

Flow networks

- I will be following CLRS, second edition for the basic definitions and results concerning the computation of max flows.

- We will depart from the usual convention and allow negative flows. While intuitively this may not seem so natural, it does simplify the development.

- The DPV, KT and CLRS (third edition) texts use the more standard convention of just having non-negative flows.

Definition

A flow network (more suggestive to say a capacity network) is a tuple $F = (G, s, t, c)$ where

- $G = (V, E)$ is a "bidirectional graph"
- the source s and the terminal t are nodes in V
- the capacity $c : E \to \mathbb{R}^{\geq 0}$

What is a flow?

A flow is a function $f : E \to \mathbb{R}$ satisfying the following properties:

1. Capacity constraint: for all $(u, v) \in E$,

$$f(u, v) \leq c(u, v)$$

2. Skew symmetry: for all $(u, v) \in E$,

$$f(u, v) = -f(v, u)$$

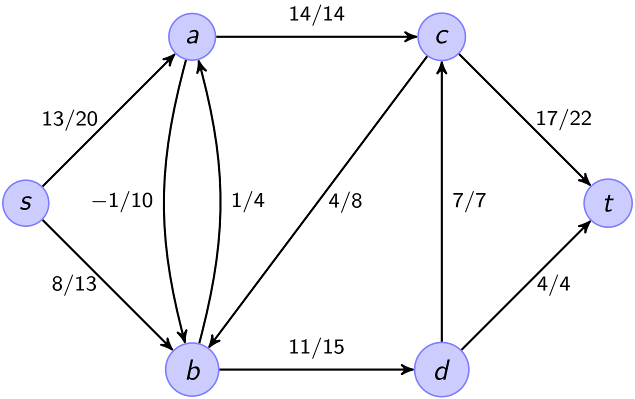
3. Flow conservation: for all nodes u (except for s and t),

$$\sum_{v \in N(u)} f(u, v) = 0$$

Note

Condition (2) and (3) are equivalent to the "flow in $=$ flow out" constraint if we were using the convention of only having non-negative flows in one direction. .

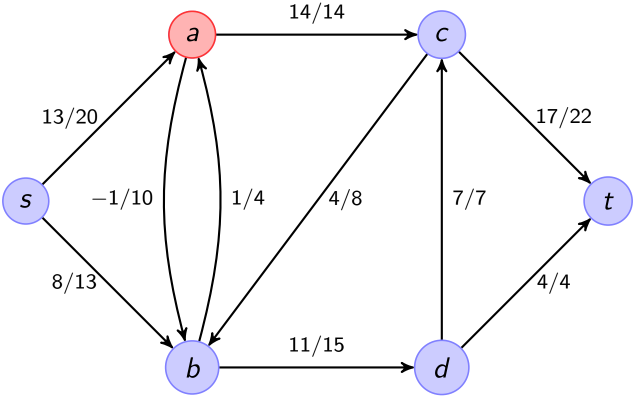
An example



The notation x/y on an edge (u, v) means

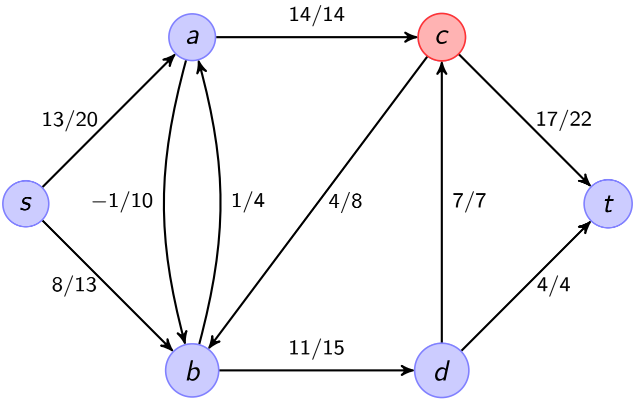
- x is the flow, i.e. $x = f(u, v)$
- y is the capacity, i.e. $y = c(u, v)$

An example of flow conservation



- For node a: $\quad f(a,s) + f(a,b) + f(a,c) = -13 + (-1) + 14 = 0$

An example of flow conservation



- For node a: $f(a,s) + f(a,b) + f(a,c) = -13 + (-1) + 14 = 0$
- For node c:

$$f(c,a) + f(c,b) + f(c,d) + f(c,t) = -14 + 4 + (-7) + 17 = 0$$

The max flow problem

The max flow problem

Given a flow network, the goal is to find a valid flow that maximizes the flow out of the source node s.

- As we will see this is also equivalent to maximizing the flow into the terminal node t. (This should not be surprising as flow conservation dictates that no flow is being stored in the other nodes.)
- We let $val(f)$ denote the flow out of the source s for a given flow f.
- We will study the Ford-Fulkerson augmenting path scheme for computing an optimal flow.
- I am calling it a "scheme" as there are many ways to instantiate this scheme although I don't view it as a general "paradigm" in the way I view (say) greedy and DP algorithms.

So why study Ford-Fulkerson?

- Why do we study the Ford-Fulkerson scheme if it is not a very generic algorithmic approach?
- As in DPV text, max flow problem can also be represented as a linear program (LP) and all LPs can be solved in polynomial time.
- I view Ford-Fulkerson and augmenting paths as an important example of a local search algorithm although unlike most local search algorithms we obtain an optimal solution.
- The topic of max flow (and various generalizations) is important because of its immediate application and many applications of max flow type problems to other problems (e.g. max bipartite matching).
    - That is many problems can be polynomial time transformed/reduced to max flow (or one of its generalizations).
    - One might refer to all these applications as "flow based methods".

A flow f and its residual graph

- Given any flow f for a flow network $F = (G, s, t, c)$, we define the residual graph $G_f = (V, E_f)$, where
  - V is the set of vertices of the original flow network F
  - E_f is the set of all edges e having positive residual capacity

$$c_f(e) = c(e) - f(e) > 0.$$

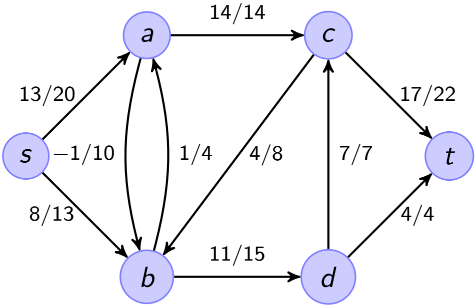
- Note that $c(e) - f(e) \geq 0$ for all edges by the capacity constraint.

Note

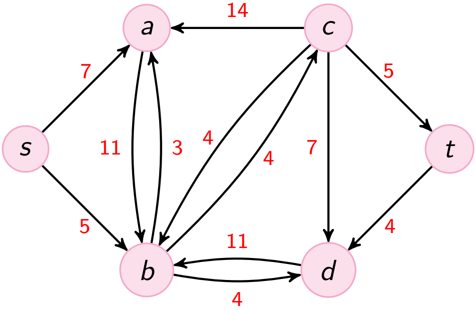
With our convention of negative flows, even a zero capacity edge (in G) can have residual capacity.

- The basic concept underlying the Ford-Fulkerson algorithm is an **augmenting path** which is an s-t path in G_f.
  - Such a path can be used to augment the current flow f to derive a better flow f'.

An example of a residual graph



The previous network flow



The residual graph

The residual capacity of an augmenting path

- Given an augmenting path π in G_f, we define its residual capacity $c_f(\pi)$ to be the

$$\min_e \{c_f(e) \mid e \in \pi\}$$

- **Note:** the residual capacity of an augmenting path is itself is greater than 0 since every edge in the path has positive residual capacity.

- **Question:** How would we compute an augmenting path of maximum residual capacity?

Using an augmenting path to improve the flow

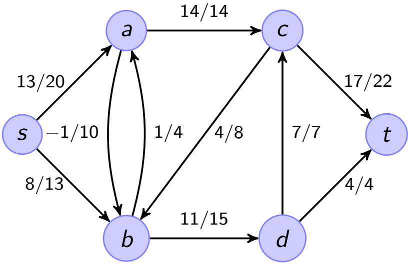
- We can think of an augmenting path as defining a flow f_π (in the "residual network"):

$$f_\pi(u,v) = \begin{cases} c_f(\pi) & \text{if } (u,v) \in \pi \\ -c_f(\pi) & \text{if } (v,u) \in \pi \\ 0 & \text{otherwise} \end{cases}$$

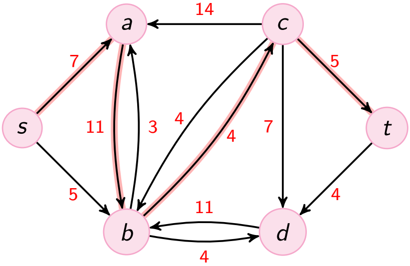
Claim

$f' = f + f_\pi$ is a flow in F and $val(f') > val(f)$

Deriving a better flow using an augmenting path

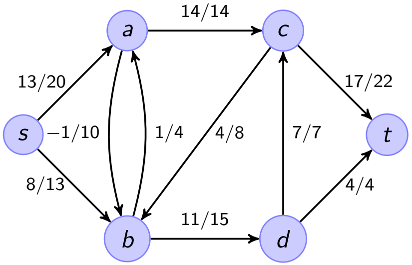


The original network flow

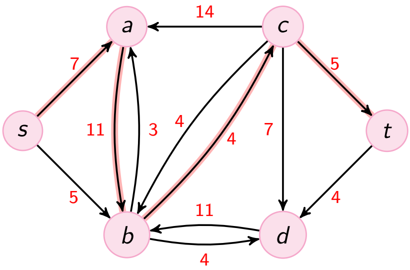


An augmenting path π with $c_f(\pi) = 4$

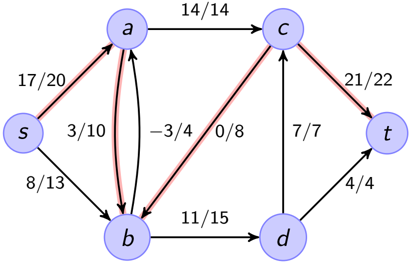
Deriving a better flow using an augmenting path



The original network flow

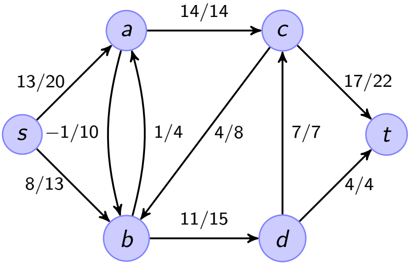


An augmenting path π with $c_f(\pi) = 4$

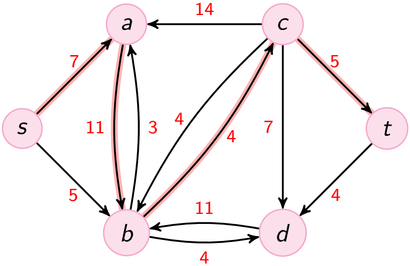


The updated flow whose value = 25

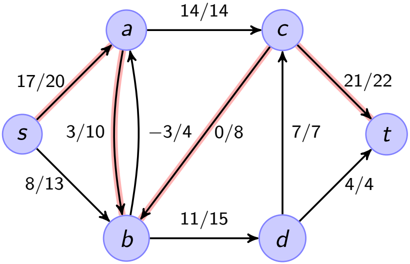
Deriving a better flow using an augmenting path



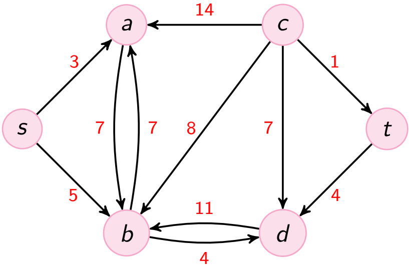
The original network flow



An augmenting path π with $c_f(\pi) = 4$



The updated flow whose value = 25



Updated res. graph with no aug. path

The Ford-Fulkerson scheme

The Ford-Fulkerson scheme

/ Initialize */*

$f := 0;\ G_f := G$

WHILE there is an augmenting path π in G_f

  $f := f + f_\pi$

    / Note this also changes G_f */*

ENDWHILE

Note

I call this a "scheme" rather than an algorithm since we haven't said how one chooses an augmenting path (as there can be many such paths)

Ford Fulkerson as a local search

- Local search is one of the most popular approaches for solving search and optimization problems.
- Local search is often considerd to be a "heuristic" since local search algorithms are often not analyzed but seem to often produce good results.
- For both search (i.e finding any feasible solution) and optimization, local search algorithms define some local neighborhood of a (partial) solution S, which we will denote as $Nbhd(S)$

The local search meta-algorithm

The local search meta-algorithm

Initialize S
WHILE there is a "better" solution $S' \in Nbhd(S)$
    $S := S'$
ENDWHILE

- Here "better" can mean different things.
  - For a search problem, it can mean "closer" to being feasible.
  - For an optimization problem it usually means being an improved solution.
- There are many variations of local search and we will hopefully study local search later but for now we just wish to observe the sense in which Ford-Fulkerson can be seen as a local search algorithm.
  - We start with a trivial initial solution.
  - We define the local neighbourhood of a flow f to be all flows f' defined by adding the flow of an augmenting path f_π to f.

Many issues concerning local search

- How do we define the local neighbourhood and how do we choose an $S' \in Nbhd(S)$?
- Can we guarantee that a local search algorithm will terminate? And if so, how fast will the algorithm terminate?
- Upon termination how good is the local optimum that results from a local search optimization?
- How can we escape from a local optimum (assuming it is not optimal)?

Local search issues for the Ford-Fulkerson scheme

- Does it matter how we choose an augmenting path for termination and speed of termination?
- That is, does it matter how we are choosing the $S' \in Nbhd(S)$?
  - **Answer:** YES, it matters but there are good ways to choose augmenting paths so that the algorithm is poly time.
  - Note that the $Nbhd(S)$ here can be of exponential size but that is not a problem as long as we can efficiently search for solutions in the local neighbourhood.
- Upon termination how good is the flow?
  - **Answer:** The flow is an optimal flow. This will be proved by the max flow - min cut theorem.
  - Note that this is unusual in that for most local search applications a local optimum is usually not a global optimum.

The max-flow min-cut theorem

- We will accept some basic facts and look at the proof of the max-flow min-cut theorem as presented in our old CSC 364 notes.
- Amongst the consequences of this theorem, we obtain that

If any implementation of the Ford Fulkerson scheme terminates, then the resulting flow is an optimal flow.

- A cut (really an s-t cut) in a flow network is a partition (S, T) of the nodes such that $s \in S$ and $t \in T$.
- We define the capacity $c(S, T)$ of a cut as

$$\sum_{u \in S \text{ and } v \in T} c(u, v)$$

- We define the flow $f(S, T)$ across a cut as

$$\sum_{u \in S \text{ and } v \in T} f(u, v)$$

Max-flow min-cut continued

Some easy facts

- One basic fact that intuitively should be clear is that

$$f(S, T) \leq c(S, T)$$

  for all cuts (S, T) (by the capacity constraint for each edge).

- And it should also be intuitively clear that $f(S, T) = val(f)$ for any cut (S, T) (by flow conservation at each node).

- Hence for any flow f, $val(f) \leq c(S, T)$ for every cut (S, T).

The max-flow min-cut theorem

The following three statements are equivalent:

1. f is a max-flow
2. There are no augmenting paths w.r.t. flow f (i.e. no s-t path in G_f)
3. There exists some cut (S, T) satisfying $val(f) = c(S, T)$
   - Hence this cut (S, T) must be a min (capacity) cut since $val(f) \leq c(S, T)$ for all cuts.

Note

The name follows from the fact that the value of a max-flow $=$ the capacity of a min-cut

The proof outline

1. (1) $\Rightarrow$ (2) If there is an augmenting path (w.r.t. f), then f can be increased and hence not optimal.

The proof outline

1. $(1) \Rightarrow (2)$ If there is an augmenting path (w.r.t. f), then f can be increased and hence not optimal.

2. $(2) \Rightarrow (3)$ Consider the set S of all the nodes reachable from s in the residual graph G_f.
    - Note that t cannot be in S and hence $(S, T) = (S, V - S)$ is a cut.
    - We also have $c(S, T) = val(f)$ since $f(u, v) = c(u, v)$ for all edges (u, v) with $u \in S$ and $v \in T$.

The proof outline

1. $(1) \Rightarrow (2)$ If there is an augmenting path (w.r.t. f), then f can be increased and hence not optimal.

2. $(2) \Rightarrow (3)$ Consider the set S of all the nodes reachable from s in the residual graph G_f.
   - Note that t cannot be in S and hence $(S, T) = (S, V - S)$ is a cut.
   - We also have $c(S, T) = val(f)$ since $f(u, v) = c(u, v)$ for all edges (u, v) with $u \in S$ and $v \in T$.

3. $(3) \Rightarrow (1)$ Let f' be an arbitrary flow. We know $val(f') \leq c(S, T)$ for any cut (S, T) and hence $val(f') \leq val(f)$ for the cut constructed in (2).

A consequence of the max-flow min-cut theorem

Corollary

If all capacities are integral (or rational), then any implementation of the Ford-Fulkerson algorithm will terminate with an optimal integral max flow.

Rational capacities

Why does the claim about integral capacities imply the same for rational capacities?

The runtime of Ford-Fulkerson

Observation

Each augmenting path has residual capacity at least one.

- The max-flow min-cut theorem along with the above observation ensures that with integral capacities, Ford-Fulkerson must always terminate and the number of iterations is at most:

$$C = \text{the sum of edge capacities leaving } s.$$

Notes

- There are bad ways to choose augmenting paths such that with irrational capacities, the Ford-Fulkerson algorithm will not terminate.
- However, even with integral capacities, there are bad ways to choose augmenting paths so that the Ford-Fulkerson algorithm does not terminate in polynomial time.

Bad example for naive Ford-Fulkerson

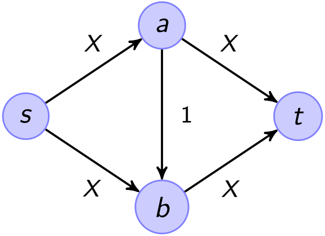


Figure: The numbers denote the capacities of the edges.

- The max-flow is clearly $2X$.
- A naive Ford-Fulkerson algorithm could alternate between
  - pushing a 1 unit flow along the augmenting path $s \to a \to b \to t$
  - pushing a 1 unit flow along the augmenting path $s \to b \to a \to t$
- This would resul in $2X$ iterations, which is exponential if say X is given in binary.

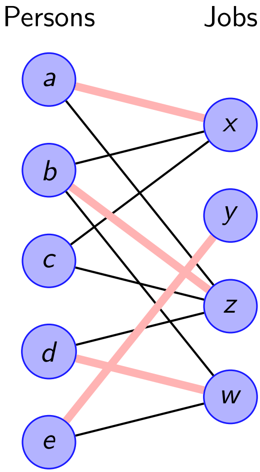
Some ways to achieve polynomial time

- Choose an augmenting path having shortest distance: This is the Edmonds-Karp method and can be found in CLRS. It has running time $O(nm^2)$, where $n = |V|$ and $m = |E|$.
- There is a "weakly polynomial time" algorithm in KT
  - Here the number of arithmetic operations depends on the length of the integral capacities.
  - It follows that always choosing the largest capacity augmenting path is at least weakly polynomial time.
- There is a history of max flow algorithms leading to a recent $O(mn)$ time algorithm (see http://tinyurl.com/bczkdfz).
- Although not the fastest, next lecture I will present Dinitz's algorithm which has runtime $O(n^2 m)$.
  - A shortest augmenting-path method based on the concept of a blocking flow in the leveled graph.
  - Has an additional advantage (i.e. an improved bipartite matching bound) beyond the somewhat better running time of Edmonds-Karp.

An application of max-flow: the maximum bipartite matching problem

The maximum bipartite matching problem

- Given a bipartite graph $G = (V, E)$ where
  - $V = V_1 \cup V_2$ and $V_1 \cap V_2 = \varnothing$
  - $E \subseteq V_1 \times V_2$
- **Goal:** Find a maximum size matching.



- We do not know any efficient DP or greedy optimal algorithm for solving this problem.
- But we can efficiently reduce this problem to the max-flow problem.

The reduction

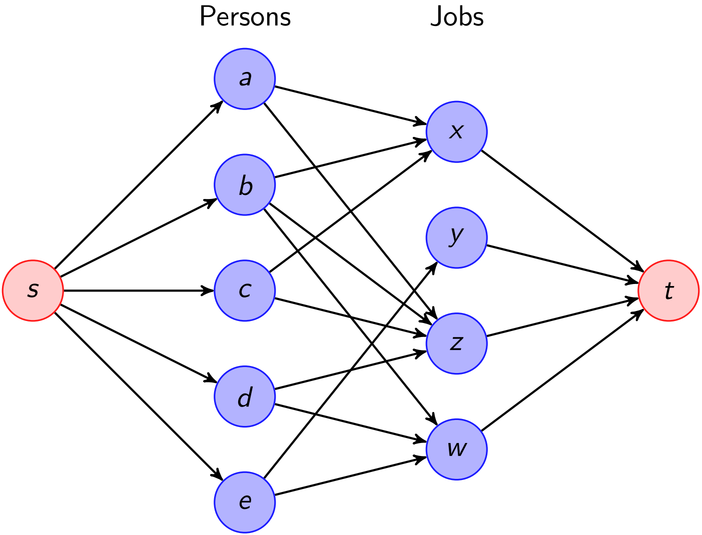


Figure: Assign every edge of the network flow a capacity 1.

The reduction preserves solutions

Claims

1. Every matching M in G gives rise to an integral flow f_M in the newly constructed flow network F_G with $val(f_M) = |M|$
2. Conversely every integral flow f in F_G gives rise to a matching M_f in G with $|M_f| = val(f)$.

Let $m = |E|, n = |V|$

- Time complexity: $O(mn)$ using any Ford Fulkerson algorithm since the max flow is at most n and $C = n$ since all edge capacities are integral and set to 1.
- Dinitz's algorithm can be used to obtain a runtime $O(m\sqrt{n})$.

A few more comments on this reduction

- When we get to our next big topic (NP completeness), we will be focusing on decision problems and as a decision problem we have $|M| \geq k$ iff $val(f_M) \geq k$.
- The reduction we are using is very efficient (linear time in the representation of the graph) and it is a special type of polynomial time reduction which we will call a polynomial time transformation.

Alternating and augmenting paths in graphs

There is a graph theoretic concept of an augmenting path relative to a matching (in an arbitrary graph).

- An alternating path π relative to a matching M is one whose edges alternate between edges in M and edges not in M.
- An augmenting path is an alternating path that starts and ends with an edge not in M.

- The reduction provides a 1-1 correspondence between augmenting paths in the bipartite G wrt. M_f and augmenting paths in G_{f_M}.

Can this reduction be extended to a maximum edge weighted matching

- In the weighted bipartite matching bipartite matching problem, we are given an edge weighted bipartite graph $G = (V, E)$ with $V = V_1 \cup V_2$ (a disjoint partition) and with say integral weights $w : E \rightarrow \mathbb{N}$
- Goal: Compute a matching M so as to maximize $\sum_{e \in M} w(e)$.
- A "reaasonable" idea is to extend the unweighted reduction by again forming a flow network with distinguished source s and terminal t nodes.
- We could then set the capacities of the edges as follows:
  - $c(x, y) = w(x, y)$ for all $(x, y) \in E$
  - $c(s, x) = \max_y \{w(x, y) : x \in V_1\}$
  - $c(y, t) = \max_x \{w(x, y) : y \in V_2\}$
- Why doesn't this work?

Edge disjoint paths: another similar max flow application

- A problem of interest in fault tolerant networks is to ensure that there are sufficiently many edge disjoint paths between any two given nodes.
- Given a directed graph $G = (V, E)$ with distinguished source s and terminal t nodes, the **goal** is to compute the the maximum number of edge disjoint paths from s to t.
- Similar to the bipartite matching transformation, we view G as a flow network $\mathcal{F}_G$ by setting the capacity of all edges equal to 1.
- Once again, because of integrality and unit capacities, we can argue that there are k edge-disjoint paths in G iff $\mathcal{F}_G$ has max (integral) flow k.
- The max flow-min cut theorem implies Menger's theorem which states that the maximum number of edge-disjoint $s - t$ paths in a directed graph is equal to the minimum number of edges in an $s - t$ cut.
- The same theorem holds for undirected graphs.

Another application of max flow-min cut

We consider an application of min cuts. The problem we wish to consider is the following binary classification problem where we want to claasify (label) vertices by one of two labels, say a and b.

We are given an edge and vertex labelled graph $G = (V, E, p, \lambda)$ where $p : E \rightarrow \mathbb{R}^{\geq 0}$, and $\lambda : V \times \{a, b\} \rightarrow \mathbb{R}^{\geq 0}$.

Our goal is to define a labeling $\ell : V \rightarrow \{a, b\}$ so that $\ell(v_i)$ is the label given to vertex v_i.

We interpret $a_i = \lambda(v_i, a)$ (resp. $b_i = \lambda(v_i, b)$) as the benefit we gain from labeling node v_i as a (resp. the benefit by labelling v_i as b). We interpret p_{ij} as the cost of labeling v_i and v_j by different labels.

In fact, I am just now discussing section 7.10 of the text which is titled image segmentation which is the problem of classifying pixels (i.e., nodes) in a grid graph where the labels correspond to foregound and background in an image. However, when reading this section, there is nothing special about grid graphs or the interpretation of the edge and vertex labels.

The objective of the binary classification problem

We want to label the vertices and lets call A, the set of nodes labelled as a and B, the set of nodes labeled b.
The goal is to maximize the following objective:

$$q(A,B) = \sum_{v_i \in A} a_i + \sum_{v_i \in B} b_i - \sum_{(i.j):(\ell(v_i) \neq \ell(v_j)} p_{ij}$$

We want to eventually restate this problem as a minimization problem.
We first restate the problem in the following way:
Let $Q = \sum_i (a_i + b_i)$. Then the equivalent objective is to maximize

$$Q(A,B) = Q - \sum_{v_i \in A} b_i - \sum_{v_i \in B} a_i - \sum_{(i.j):\ell(v_i) \neq \ell(v_j)} p_{ij}$$

The is equivalent then to minimizing:

$$q'(A,B) = \sum_{v_i \in A} b_i + \sum_{v_i \in B} a_i + \sum_{(i.j):\ell(v_i) \neq \ell(v_j)} p_{ij}$$

Minimizing q' as a min cut problem

This is similar to the way we reduced (or transformed) maximizing the size of a matching in a bipartite graph to computing a maximum flow in a related flow network.

Namely, we want change the graph into a network flow graph. Each edge will become two directed edges and we will have new source s and target t nodes where s (resp t) will now be thought of as super node representing nodes labeled as a (resp. as a super node representing nodes labeled b).

We will place capacities between the source s and other nodes to reflect the cost of a "mislabel" and similarly for the terminal t.

The min cut will then correspond to a min cost labelling. We construct the flow network $\mathcal{F} = (G', s, t, c)$ such that

- $G' = (V', E')$
- $V' = V \cup \{s, t\}$
- $E' = \{(u, v) | u \neq v \in V\} \cup \{(s, u) | u \in V\} \cup \{(u, t) | u \in V\}$
- $c(i, j) = c(j, i) = p_{i,j}; c(s, i) = a_i; c(i, t) = b_i$

Depicting the *transformation* of the binary labeling problem to the min cut problem

Consider min cut (A, B) in G'.

- A = foreground.

$$cap(A,B) \;=\; \sum_{j \in B} a_j \;+\; \sum_{i \in A} b_i \;+\; \sum_{\substack{(i,j) \in E \\ i \in A,\ j \in B}} p_{ij}$$

if i and j on different sides, p_{ij} counted exactly once

- Precisely the quantity we want to minimize.

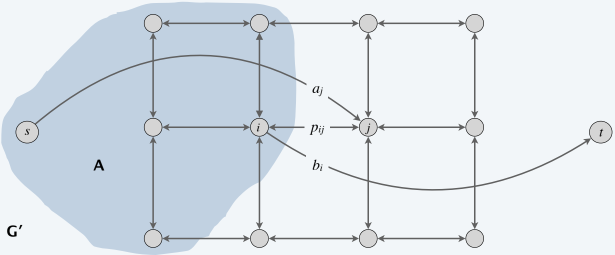


Figure: Figure taken from Kevin Wayne's slides

The more general metric labeling problem

The binary classification problem is a special case of the following more general problem called metric labeling, later considered in the section 12.6 of the text when discusasing approximation algorithms.

- The input is an edge weighted graph $G = (V, E)$, a set of labels $L = \{a_1, \ldots, a_r\}$ in a metric space with distance metric d, and functions $w : E \to \mathbb{R}^{\geq 0}$ and $\lambda : V \times L \to \mathbb{R}^{\geq 0}$.
- We want to construct a labeling $\ell : V \to L$
- $\beta(u, a_j)$ is the benefit of giving label a_j to node u, and d represents the distance between labels.
- Goal: Find a labelling $\lambda : V \to L$ of the nodes so as to maximize $\sum_u \beta(u, \lambda(u)) - \sum_{(u,v) \in E} w(u, v) \cdot d((\ell(u), \ell(v))$
- For example, the nodes might represent documents, the labels are topics, and the edges are links between documents weighted by the importance of the link.
- When there are 3 or more labels, the problem is NP-hard even for the case of the $\{0,1\}$ metric d where $d(a_i, a_j) = 1$ for $a_i \neq a_j$.