MF5: Max Flow Variant – Demands

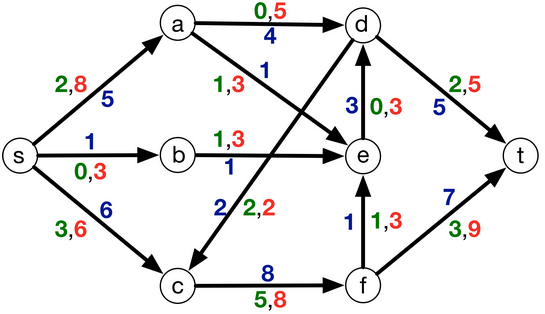
Notes for CS-8803-GA: Introduction to Graduate Algorithms

Georgia Tech (Dr. Eric Vigoda), Fall 2017

as recorded by Brent Wagenseller

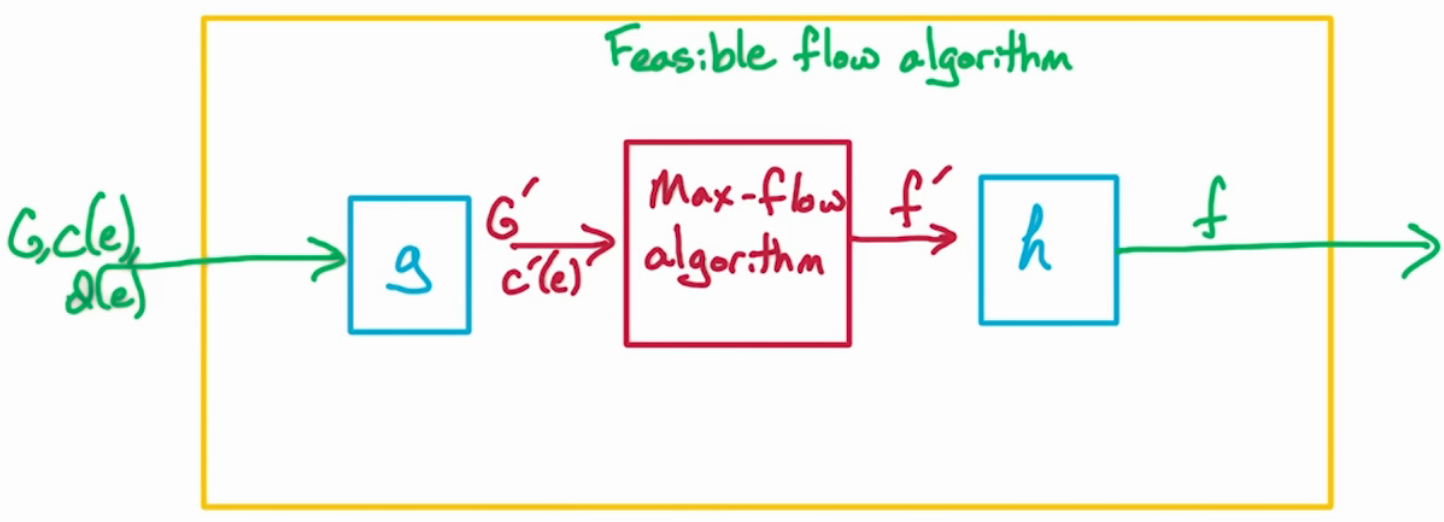
Max-Flow With Demands

* We are going to generalize the max-flow problem by adding in demand constraints along every edge
  + We will reduce this generalization to the standard max-flow problem
* The input
  + Mostly same as before
    - Directed graph G=(V, E) with s,t ∈ V
    - Capacities c€ > = for e ∈ E
  + We now also have a demand d
    - d€ >=0 for e ∈ E
* Our goal here is to find a feasible flow
  + A **feasible flow** is a flow f where: for e ∈ E, d(e) <= f(e) <= c(e)
    - In other words, all flows are capped at capacity (as before) but ALSO are lower bounded by the demand d(e)
    - Of course, we still want to maximize the flow; THAT SAID, the trouble is making sure all d(e) are satisfied
    - There may not actually be a feasible flow that satisfies the lower bound demands!
    - If we can find a feasible flow, it will be easy to maximize that flow
      * Note that **A** feasible flow may not be the max flow for the graph, it can only maximize the graph that satisfies the demand!
* Example:



* + Green are demands
  + Reds are capacities
  + Blues are flow\* values

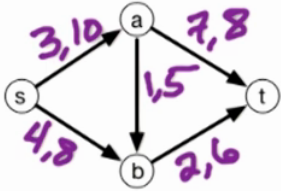
Reduction from Feasible Flow → Max Flow



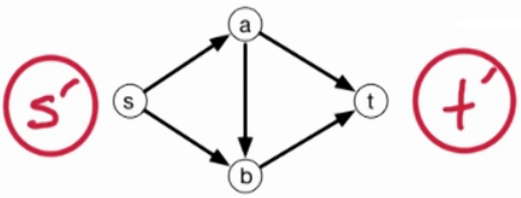
* To do this, we will treat the max-flow algorithm as a black-box, with an input G` and we get out f`
* We will use the output from the max-flow to build the feasible flow algorithm
* The initial inputs to the feasible flow problem
  + G(V, E)
  + Constraints c(e)
  + Demand d(e)
* This will feed into a function ‘g’, which will change the original input to an acceptable input for the max-flow algorithm
  + g transforms G to G` and c(e) to c`(e)
    - This is important, as these need to be altered to accommodate the feasible flow as d(e) is NOT an input to the max-flow algorithm
    - g effectively encodes the demands d(e) into the graph for it to be run as a regular max-flow problem
    - G` is input that is acceptable for the regular max-flow problem
* The max-flow algorithm runs as a black-box
  + This will give an f` that must once again modify the results to align with the original feasible flow problem
    - This is done by a function known as “h”
    - h transforms flow f` into a feasible flow f, which satisfies the original input
      * f` is output that satisfies the regular max-flow problem, and f is output that satisfies the feasible flow problem
  + g transforms the input to a regular max-flow, and h transforms the solution to a feasible flow solution

Reduction: Vertices

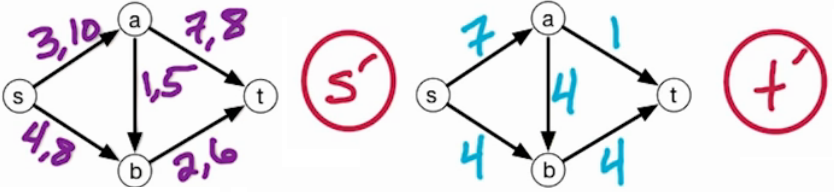
* The first thing that must be done is to construct G`, c`(e) that can be used in the max-flow algorithm
  + Note that we will be encoding the demand d(e) into the capacity c(e); the new capacity will be c`(e)
* Changing G to G` and c(e) to c`(e)
  + Our example graph:



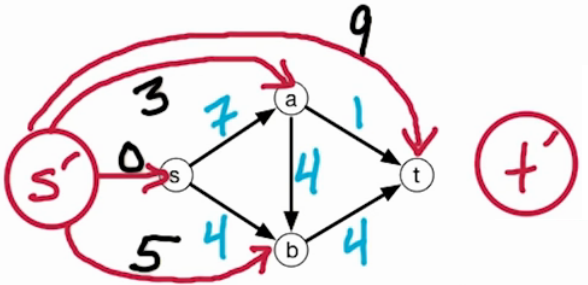
* + - This is the initial G and c(e)
    - Note that the first number is the demand, and the second is the capacity
  + The first thing we do is add S` and T`, which are new versions of s and t:



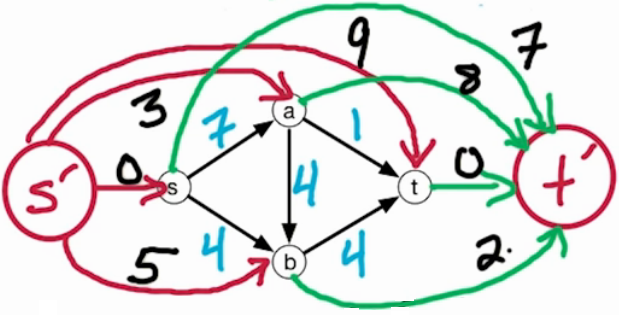
* + - The old s, t remain in the graph but now they are just internal vertices
  + Edges
    - The edges themselves remain, but the capacities will be modified to accommodate the demand
    - Intuition
      * We want to capture if there is nonnegative flow across an edge if and only if we can construct a flow in the original network where the flow is at LEAST the demand
      * The flow in the new graph G’ satisfies the new capacity constraint if and only if the flow f satisfies the capacity constraint in the original network
    - Actual way to factor in demand into the capacity – shift c by d:
      * C`(e) = c(e) – d(e)
    - Result (original G on left, G` on right after factoring in demand into capacity):



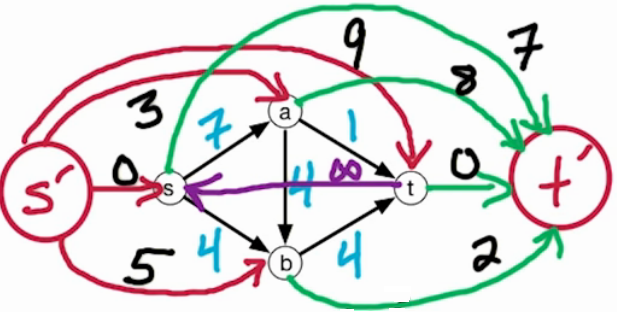
* + In the left (original) graph, d(e) is listed first, then c(e) for all edges
  + Intuitively, this works because c(e), d(e), and c`(e) = (c(e) - d(e)) Are ALL >= 0; since this is true, we can natively ‘bake-in’ the demand; even of an edge has capacity 0 in G` we can re-add d(e) in G so long a G` has a saturating flow (see what a saturating flow is later in the lesson)
  + We still need to do something else though – add more edges
    - (see above image) we need to make sure a valid flow on the right is a valid flow for the graph on the left
      * For example, edge AT requires 7 units and edge AB requires 1; this means vertex ‘a’ needs no less than 8 units as intake, so we must ensure edge SA accommodates this and at least delivers the minimum 8; the new edges will help ensure this
    - To start accomplishing this, we add an edge from s` to ALL other nodes (including s):



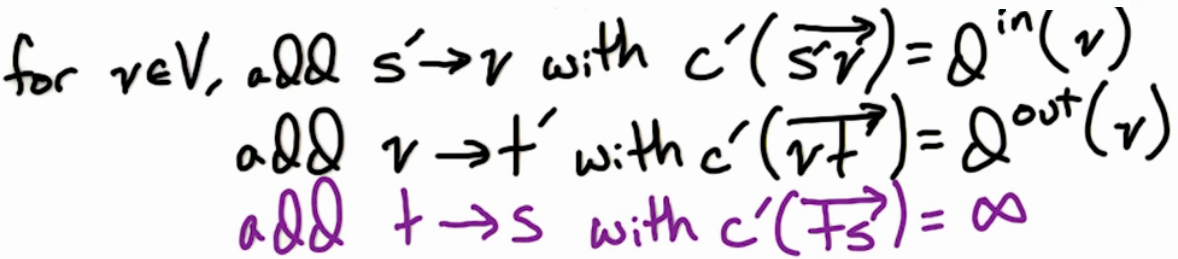
* + - * The capacities on these new edges are the TOTAL demand ENTERING for the ENTIRE vertex; for example, the demand for all incoming edges for vertex b was 4 + 1 = 5
    - We now do something similar with t`; for t`, we make an edge out of every vertex into t`, with the capacity being the TOTAL demand LEAVING the vertex:



* + - One last snag: technically nothing can enter the old s or leave the old t; to get around this, we make one final edge from t to s and have it as unlimited capacity:

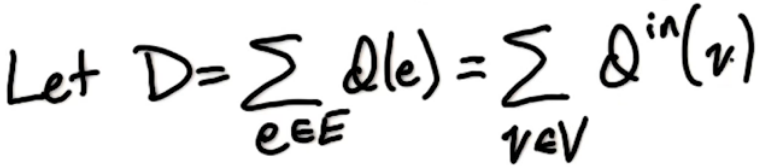


* + - What we did, in a nutshell:



Saturating Flows

* The max flow is bounded by what is coming out of s` and what is going into t`
* Let D = total demand of all out edges (note: this is the same as summing all incoming demand edges OR all outgoing edges):



* Total capacity out of s`
  + This is actually demand into each vertex, summed
  + In other words, this is the demand into all vertices, summed; this is equal to D above
* Total capacity into of t`
  + This is actually demand out of each vertex, summed
  + In other words, this is the demand out of all vertices, summed; this is also equal to D above
* Due to the capacities of s` and t`, we know for flow f` in G`, size(f`) is upper-bounded by the capacity out of s` and lower-bounded by the capacity into t`, both of which are D
  + This means the size of the flow f` is at most D; we say f` is **saturating** if size(f`) = D, which means its fully meeting its demand
    - To find this out we would have to run a network flow on G` to see if its size = D
* **Lemma**: G has a feasible flow if and only if G` has a saturating flow

Saturating ⇒ Feasible

* It needs to be proven that a saturating flow in G` means a feasible flow in G
* To do so, we take a saturating flow f` in G`
  + From this, we construct a feasible flow f in G
  + To do this, recall the capacities in G` were the capacities in G shifted by the demand d
    - Therefore, we let the flow in G = the flow in G` plus the demand
    - If we had 0 flow in G`, then we have the min flow possible in G
    - Therefore,

f(e) = f`(e) + d(e)

* From here we need to check two things:
  + f is valid and does not exceed capacity
  + f is feasible and meets at least its corresponding d
  + <both of these are proven in the lecture>
* f is valid
  + For every internal vertex, we need to show that the flow in = the flow out (except for s and t)
  + The main point is in of f` is the out of f` for every vertex, so this should parlay into f
  + <this was proven in the lecture>

Feasible ⇒ Saturating

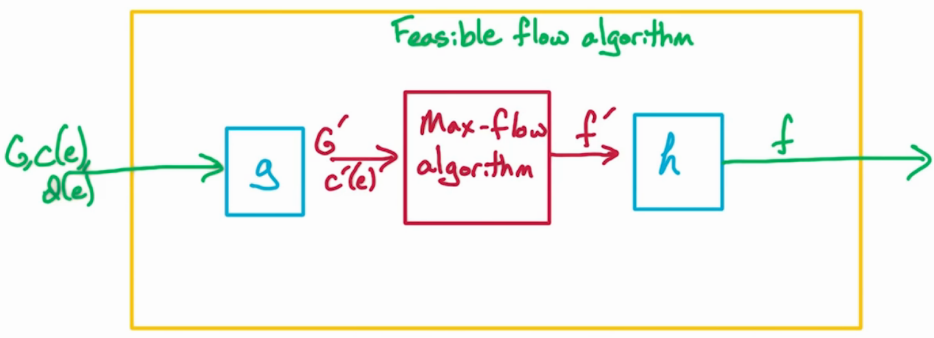
* <This was also proven in the lecture>

f` constraints

* We need to check that the flow into every vertex v is equal to the outflow in v (for G`)
* <this is proven in the lecture>

Max Feasible Flow

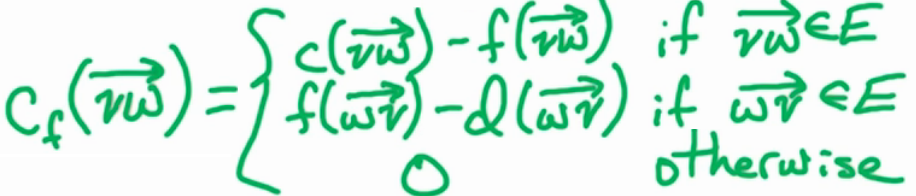
* Using the image used previously:



* We now have f`, which is an input to h
* What happens in “h”?
  + We now run a modified max flow algorithm on our graph
    - HOWEVER, instead of initializing all of our f(e) values to 0, we initialize to

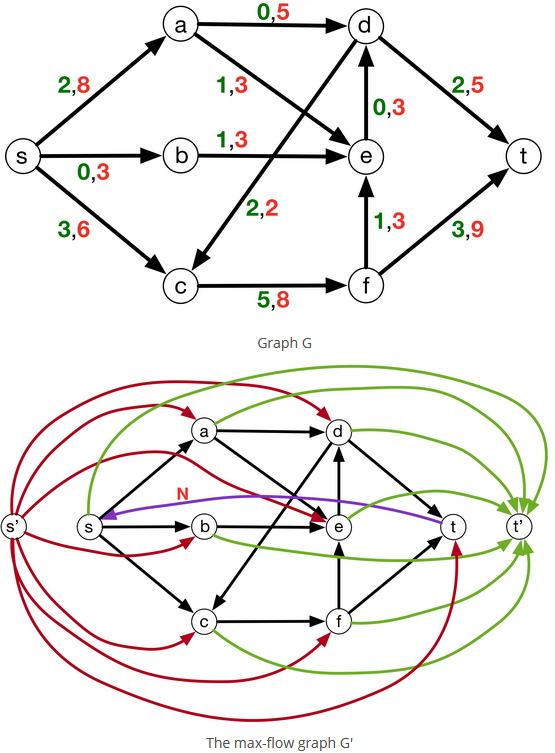
f(e) = f`(e) + d(e)

* + We now run our modified max-flow algorithm on the graph and we get a max-flow that satisfies the demand!
    - Why is it modified? Because we cannot allow the traversal of residual edges to shrink any edge f(e) to under the value of d(e); if this happens we break our demand promises!
    - So the capacity for the forward edges will be similar to before, the capacity – current flow
    - For backedges, we can only decrease by how much the flow exceeds demand; in other words, f(e) – d(e)
    - Mathematically,



Capacity Quiz

* Find the capacities from s` to all vertices, and all vertices to t` in G` below:



Answers:

capacity = { ("s'", "a"): 2,

("s'", "b"): 0,

("s'", "c"): 5,

("s'", "d"): 0,

("s'", "e"): 3,

("s'", "f"): 5,

("s'", "t"): 5,

("a", "t'"): 1,

("b", "t'"): 1,

("c", "t'"): 5,

("d", "t'"): 4,

("e", "t'"): 0,

("f", "t'"): 4,

("s", "t'"): 5 }