CS 7301.003.20F Lecture 12—September 28, 2020

Main topics are #maximum_flow in #planar_graphs.

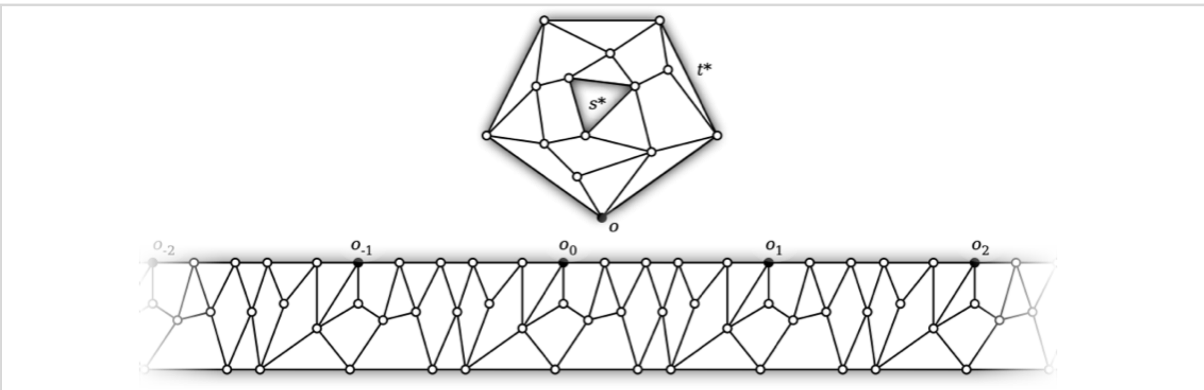
Maximum Flow Continued

- Let's pick up from where we left off last time.
- We're given a plane graph G = (V, D) with two designated vertices s and t along with a capacity function c : D → R^+ on the darts. We want to compute the maximum s,t-flow.
- Venkatesan ['87] shows how to find an s,t-flow of value lambda > 0 if one exists:
  - Let P be an arbitrary s,t-path.Let pi(d) =
    - 1 if d in P
    - -1 if d in rev(P)
    - 0 otherwise
  - Define the residual capacity of a dart as c_lambda(u → v) = c(u → v) - lambda * pi(u → v). Let G^*_lambda denote the *dual residual network* that treats these residual capacities as dart lengths.
  - An s,t-flow of value lambda exists if and only if G^*_lambda has no negative cycle.
  - Suppose there isn't one. Let dist_lambda(f^ *) be the distance from o to f^ * in G^ * with regard to c_lambda and define slack_lambda(p → q) := dist_lambda(p) + c_lambda(p → q) - dist_lambda(q).
  - Define phi_lambda(p → q)
      - := dist_lambda(q) - dist_lambda(p) + lambda * pi(p → q)
      - = c(p → q) - slack_lambda(p → q).
- How do we find the maximum good value for lambda?
- We'll use a strategy similar to last Monday's. We'll start with lambda = 0 and compute the dual shortest path tree. Then we'll continuously increase lambda as we compute so-called *parametric shortest paths*.
- dist_lambda and phi_lambda will vary continuously as we increase lambda, but shortest path tree T_lambda will change at discrete *pivots* just like before.
- And just like before, we'll wait until just before some slack_lambda(p → q) < 0, pivot p → q into T_lambda, and pivot pred(q) → q out.
- Define slack'_lambda(p → q) := derivative of slack for p → q and path_lambda(p) := the shortest path in T_lambda to p
- slack'_lambda(p → q)
  - = dist'_lambda(p) + pi(p → q) - dist'_lambda(q)
  - = -pi(path_lambda(p)) - pi(p → q) - pi(rev(path_lambda(q)))
  - = -pi(cycle(T_lambda, p → q))
  - in {-1, 0, 1}

- Also, we see slack'_lambda(p → q) = -slack'_lambda(rev(p → q))
- We'll call p → q *active* if slack'(p → q) = -1.
- But how do we find the active darts?
- Let L_lambda = (G \ T_lambda)^*, the complementary spanning tree of G.
- Assuming shortest paths are unique, L_lambda is the set of *loose* edges, those where both darts have slack > 0.
- LP_lambda := unique path from s to t in L_lambda
- Lemma: d^* is active if and only if d in LP_lambda
  - Darts of T_lambda have slack = 0, so slack' = 0 for them and their reversals.
  - d^* is active if and only if pi(cycle(T_lambda, d^*)) = 1 if and only if C(d) = (cycle(T_lambda, d^ *))^ * is an s,t-cut
  - If d^* is active, then LP_lambda contains at least one edge of C(d). But d is on the only loose edge of C(d), so d in LP_lambda
  - If d in LP_lambda, then C(d) is an s,t-cut.
- So now the algorithm behaves similar to MSSP.
- As we increase lambda, slack_lambda(d) decreases for all d in LP_lambda and increases for all d in rev(LP_lambda).
  - Equivalently, phi_lambda(d) increases along LP_lambda and decreases along rev(LP_lambda).
  - It's like we're pushing flow from s to t along LP_lambda.
- Pivot d into T_lambda when slack_lambda(d) = 0. It's like we saturated the dart.
- Pivot pred(q) → q out to T_lambda at the same time. It's like we made a new augmenting path!
- Eventually, we do a pivot that creates a directed cycle of slack 0 darts.
- But that means each of those darts is saturated. We found the minimum s,t-cut and the maximum s,t-flow!
- Pseudocode:
  - Create initial dual shortest path tree T_0
  - Maintain primal spanning tree L
  - While L is connected
    - LP ← path from s to t in L
    - (p → q) ← min slack edge in LP^*
    - decrease slacks along LP by slack(p → q). Increase slacks along rev(LP) by the same amount.
    - delete (p → q)^* from L
    - insert (pred(q) → q)^* into L
    - pred(q) ← p
  - phi ← c - slack

Analysis

- Using dynamic forests, each step of the algorithm can be implemented in O(log n) time. Our running time therefore depends on the number of pivots.
- Let path_i denote the shortest walk from o to q s.t. pi(path_i(q)) = i.
- We see dist_lambda(q) = min_i (c_lambda(path_i(q)).
- We can also observe that whenever path_lambda(q) changes, pi(path_lambda(q)) increases by 1, because pi(cycle(T, p ➡ q)) = 1 if p ➡ q is pivoting in.
- So now we want to know, for which i is p ➡ q in path_i(q)?
- Imagine removing faces s^* and t^* from the plane. We now have a sphere with two boundary, also known as an *annulus*.
- We'll define something called the *universal cover* as follows: Imagine cutting along that path P from earlier, turning our annulus into a disk. Now make a doubly infinite sequence …, G^ *_-1, G^ * _0, G^ *_1, G^ *_2, … of copies of G^ * and paste them together along them respective copies of P.
- Formally, its a plane graph Gbar* = (Vbar*, Ebar *) where Vbar * = {p_i | p in V * and i in Z} and Ebar * = {p_i ➡ q_{i + pi(p ➡ q)} | p ➡ q in E *}. We also have dart costs c(p_i ➡ q_j) = c(p ➡ q).
- Each vertex p_i is a *lift* of p to Gbar *. There is a *projection map* omega bar(Gbar * ➡ G *) that drops subscripts so omega bar(p_i) = p and omega bar (p_i ➡ q_j) = p ➡ q.
- The preimage omega bar^-1(Pi) for any path Pi in G^* is a doubly-infinite set of paths in Gbar * called the *lifts* of Pi. If Pi starts at p and ends at q, then for any i, there is a lift of Pi from p_i to q_{i + pi(Pi)}.
- s^* and t^* lift to two unbounded faces sbar * and tbar *, and every other face lists to an infinite sequence of faces.

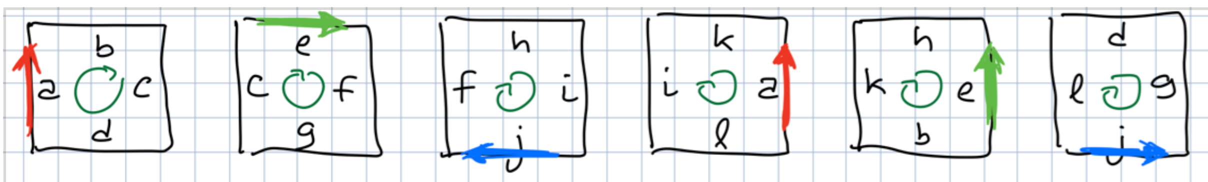


- So now for the punchline. Each path_i(q) in G^* lifts to a shortest path to q_0 from o_{-i}.
- The i for which p ➡ q is in path_i(q) are the set of i for which the shortest path from o_{-i} goes through p_{-pi(p ➡ q)} ➡ q_0.
- But we saw on Monday that these i are contiguous!
- So p ➡ q pivots into T_lambda at most once, and it leaves at most once.
- Which implies O(n) pivots.

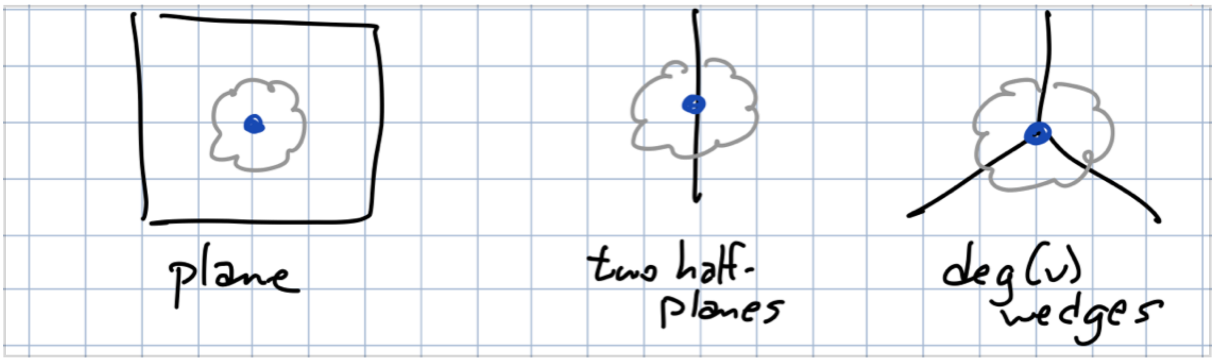
- Which implies an O(n log n) running time!
- Some quick notes:
  - Erickson ['10] described this formulation of the algorithm based on parametric shortest paths.
  - His algorithm is essentially identical to one by Borradaile and Klein ['09]. However, they describe things mostly in the primal graph, sending flow along LP each iteration to always have a "leftmost flow" of each value lambda. Their analysis is much more complicated, because they focus on how often you can saturate each dart with this primal flow.

Surface Maps

- It's finally time to leave the plane. Let's do that by going back to something we saw before.
- Recall a rotation system can be described as a triple of permutations succ, rev, and next from darts to darts (Erickson now calls these vnext, rev, and fnext so make it easier to remember which is which. Maybe we should start doing that too?)
  - rev is an involution w/o fixed points
  - fnext [i.e. next] = rev circ vnext
  - orbits of vnext go ccw around vertices at head
  - orbits of rev are edgs
  - orbits of fnext go cw around faces to right
- V - E + F = 2 in connected planar graphs, but what if they equal something else?
- We still have an embedding… it's just not planar!
- Imagine every face (orbit of fnext) as a polygon. Label the sides of these polygons with the names of distinct darts. This construction is called the *polygonal schema* of the embedding.
- Each edge appears twice on the boundary of the polygons.
- Glue the polygons together at their corresponding darts by identifying each rev(d) with the reversal of dart d.



- If you look at a sufficiently small neighborhood around each point after gluing



you'll see these neighborhoods are all homeomorphic to the plane.

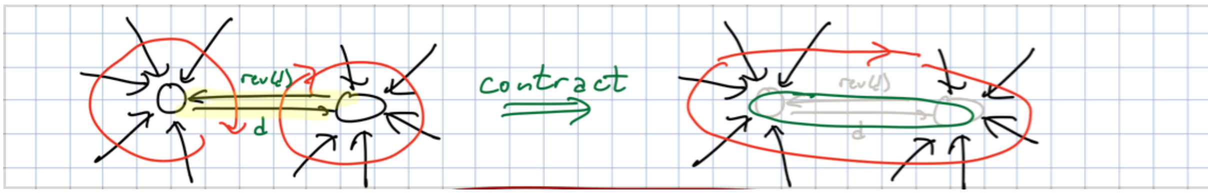
- What we've created is an orientable compact 2-manifold. Otherwise known as a *surface*.

- There are infinitely many of these things, and they're distinguished (up to homeomorphism) by their genus.
- Intuitively, the genus is the number of handles you glue onto a sphere to get the surface, but sometimes its hard to tell what the handles are.
- Formally, the *genus* is the maximum number of disjoint simple closed curves on the surface whose complement is still connected.

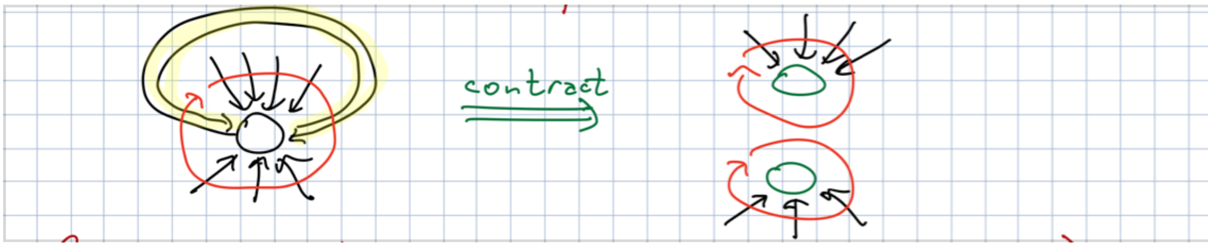
Trees, Co-trees, and Formulas

- Remember, if G is planar and T is a spanning tree, then (E \ T)^* is a spanning tree of G^*. But that's no longer true when genus > 0.
- An *isthmus* is any edge with the same face on both sides. We may have isthmuses now that are not bridges!
- We can contract any edge that's not a loop without messing with the faces. By duality, we can delete any edge separating distinct faces.
- Take any rotation system, and let g be the genus of the underlying surface.
- Suppose we contract non-loops until there's a single vertex. The V - 1 edges we contracted form a spanning tree T, but the faces remain intact.
- Now delete non-isthmuses until only one face remains. The F - 1 deleted edges form a dual spanning tree.
- But now there are L leftover edges that are all both loops *and* isthmuses. Counting the edges, we see V - E + F = 2 - L.
- These remaining edges form what is called a *system of loops*, a surface map with one vertex (the basepoint) and one face homeomorphic to an open disk.
- But how are L and g related? To find out, we're going to do something previously forbidden. We're going to contract a loop!

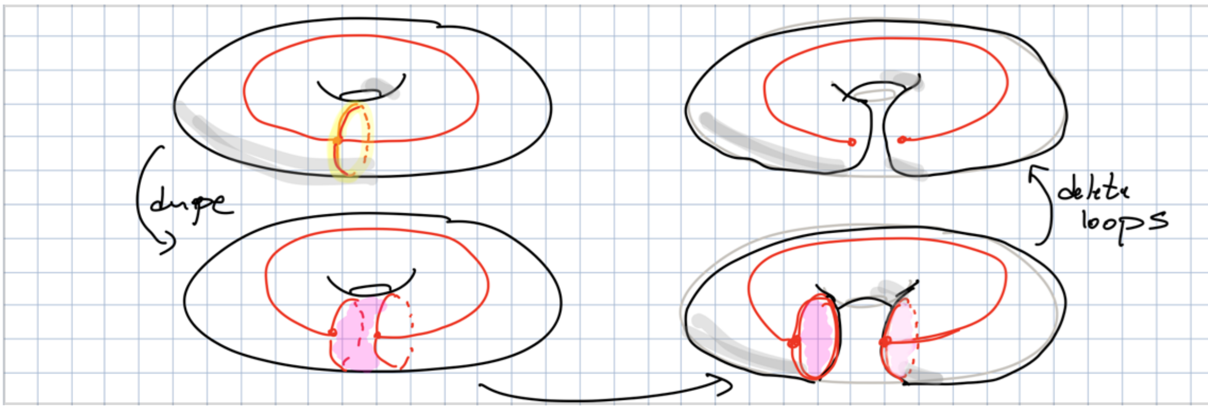
- Let's recall how to modify the rotation system to contract a non-loop edge.



- vnext(vprev(d)) ← vnext(rev(d)) and vprev(vnext(d)) ← vprev(rev(d))
- But if we do those operations to a loop, we split the vertex.



- And if we delete an isthmus, by duality we split a face.
- So what would this look like on the actual surface?
- Well, you can't get from one side of the loop to the other anymore, so it's like we cut a handle. Here's another way to imagine what happened:



- The result is that we now have genus g - 1 but still only one face.
- The face uses both vertices, so there is at least one non-loop edge. Contract it as well.
- Now we have a new system of L - 2 loops on a surface of genus g - 1.
- Now we just need to figure out some base cases, and we'll be done.
- If L = 0, then we have the trivial map of a single vertex on a sphere.
- If L = 1, then we have two darts d and rev(d).
  - So vnext(d) = rev(d), implying fnext is the identity function. But that means we have two faces. There is no system of one loop (on an orientable surface).
- So by induction, we conclude V - E + F = 2 - 2g.