CS 7301.003.20F Lecture 9—September 16, 2020

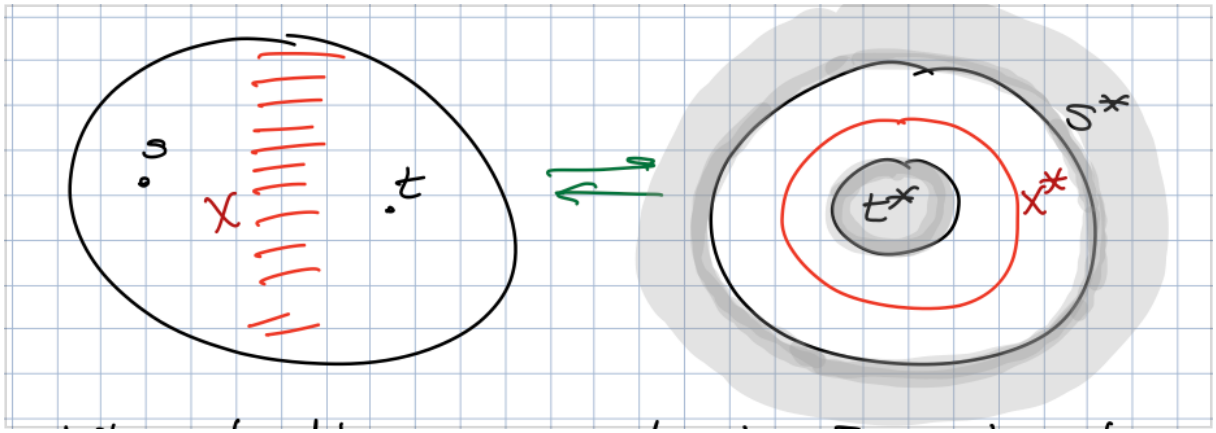
Main topics are #planar_graph_min_cut.

Minimum Cut

- One of the most heavily studied problems in algorithms is the *minimum s,t-cut problem*.
- In the undirected version of the problem, we're given a graph G = (V, E) with non-negative *capacities* c : E ➔ R_{≥ 0} on the edges along with two designated vertices s and t.
- The goal is to find an *s,t-cut*, a cut (S, T) (a bipartition of the vertices) such that s in S and t in T. The capacity of the cut total capacity of edges in the associated edge-cut: sum_{uv in E : u in S, v in T}. We want a cut of minimum capacity.
- In a previous algorithms class, you may have learned an algorithm or two to solve the more general directed version of this problem. The best algorithm taught in these classes usually runs in O(n m^2) time.
- After many decades of improvements we now know an O(mn) time algorithm. For sparse graphs where m = O(n), there's also an O(n^2 / log n) time improvement. Pretty much all algorithms for general graphs are based on first computing a *maximum s,t-flow* and then looking for a saturated bond.
- Today, we're going to see how computing a minimum cut directly is actually easier in undirected planar graphs.

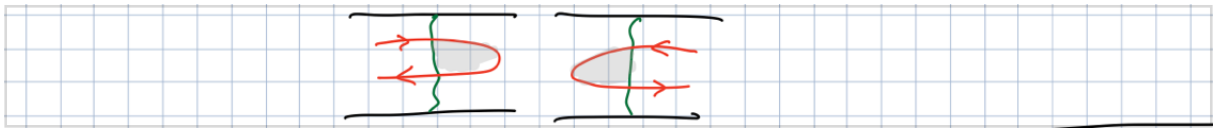
Cut-Cycle Duality

- The main thing that makes this faster algorithm possible is the duality between edge cuts and even subgraphs.
- Recall a *bond* is a minimal edge-cut. Since we're going with non-negative capacities, the minimum s,t-cut had better be a bond.
- Bonds in particular are dual to cycles.
- Notice how in the dual graph, s and t have become faces s^* and t^*. The minimum cut is a cycle that separates two sets of faces, one containing s^ * and the other containing t^ *.
- Let's embed the dual graph so s^* is the outer face.
- (The images of) cycles separating s^* from t^* are freely homotopic to one another, just slide across the dual faces that are different between two s,t-cut duals to get the homotopy.
- Therefore, if we have a path pi (in the plane) between s^* and t^*, all such cycles have a reduced crossing sequence of length 1 with pi. Our goal is to find the shortest such cycle.

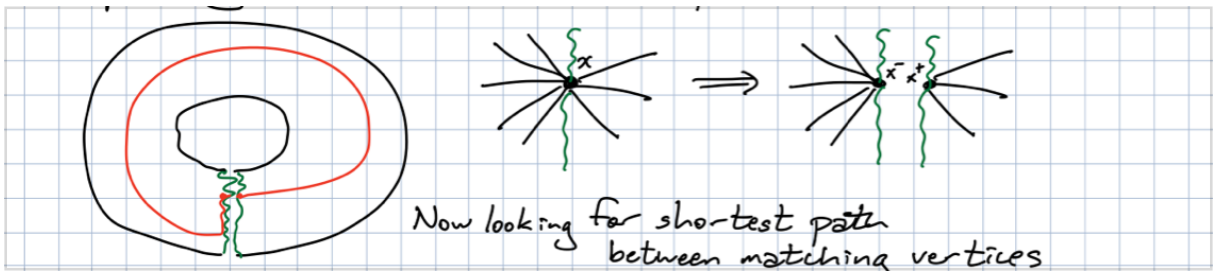


Cutting Along a Shortest Path

- There's a choice for pi that makes finding the cycle much easier.
- Let pi be the shortest path from s^* to t^*. We can find it in O(n log n) time using Dijkstra's algorithm. There's also an O(n) time shortest path algorithm by Henzinger et al. ['97] that uses separators in a fancy way.
- Claim: The (dual) minimum s,t-edge cut crosses pi exactly once.
  - If it crosses more than once, there's a bigon we can reduce.



  - The bigon is touching the (undirected) shortest path twice, so we'll just shortcut from one crossing point to the other. Inductively, we can remove all but one crossing this way.
  - This is the only part where we really need the graph to be undirected for an easy fast algorithm.
- So how do we find the shortest cycle crossing pi once?
- We cut along pi by duplicating the vertices and edges of pi and splitting the incident edges to stay on the appropriate sides.

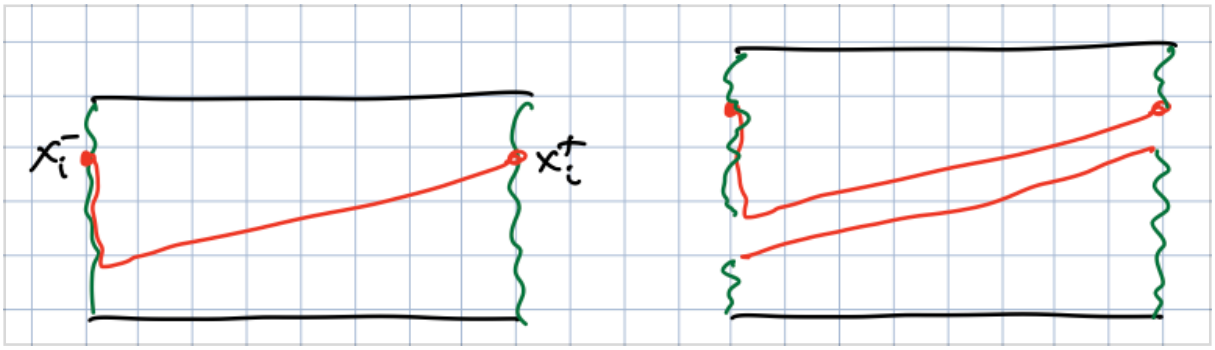


- If the shortest cycle has a crossing at some vertex x, it now looks like a path between x^- and x^+. In fact, it will be the shortest such path!
- But we don't know which x is correct ahead of time. Maybe we should try them all? Suppose pi has k vertices:
  - We could run Dijkstra k times for O(k n log n) time total.
  - Or use the linear time Henzinger et al. shortest path algorithm k times to find the

shortest cycle in O(kn) time total.

- On Monday, we'll see how to quickly find several shortest path distances from vertices rooted on a single face. This approach would take O(n log n) time.

- Reif ['83] took a different approach based on divide-and-conquer.
- First, we compute the shortest path from x^-_{k / 2} to x^+_{k / 2}.
- Then we cut along this path, replacing any degree 2 vertices that result with single edge.



- We just partitioned the faces into two sections of size n. By Euler's formula, the total complexity of the two sides is still O(n).
- A shortest path from any x^-_i to x^+_i stays entirely in one side of the partition, because we could again shortcut along the path from x^-_{k / 2} to x^+_{k / 2} it if it didn't.
- So we recursively find shortest paths on both sides.
- Using Dijkstra to find each shortest path, the total time we spend is T(n, k) = O(n log n) + T(n_1, k / 2) + T(n_2, k / 2) where n_1 + n_2 = n. This recurrence solves to O(n log n log k).
- If we use the O(n) time shortest path algorithm instead, we again get O(n log k) time total.

r-divisions and FR-Dijkstra

- When I started grad school, this was the end of the story, and some people, including my advisor, assumed O(n log n) was the best you could do in the worst case.
- But Italiano et al. made a little breakthrough in 2011 using a data structure based on recursively computed cycle separators.
- By recursively finding cycle separators for the edges, we can, for any r, partition the edges into O(n / r) pieces, with each piece having r edges and at most O(sqrt{r}) vertices that sit on its boundary incident to edges in other pieces.
- That means there's a total of O(n / r) * O(sqrt{r}) = O(n / sqrt{r}) boundary vertices across all pieces.
- A *hole* of a piece is a face of the piece that doesn't exist in the original graph.
- By alternating between computing balances vertex and hole separators, we can find an r-division with only O(1) holes per piece.
- Naively, all this recursive partitioning would take O(n) time across all separations in each recursion level, for O(n log (n / r)) time total.
- But by being more careful, we can compute it in O(n) time [Klein, Mozes, Somner '13] for

any r.

- The *dense distance graph* (DDG) for the r-division is the union of complete graphs of the boundary vertices of each piece. Each edge is given the shortest path distance between its endpoints within the one piece. Using the procedure we'll learn on Monday, we can compute the DDG in O(n log r) time total.
- While working on fast algorithms for shortest paths, Fakcharoenphol and Rao ['06] described a way to compute the shortest path through the whole DDG between any pair of boundary vertices of G^* in only O(n / sqrt{r} log n log r) time. Their algorithm is now referred to as FR-Dijkstra.
- So here's what Italiano et al. propose: find the shortest x^-_i to x^+_i paths for each of the O(n / sqrt{r}) boundary vertices x_i on pi. This is done by running Reif's divide-and-conquer approach using FR-Dijkstra. The total time taken is O(n / sqrt{r} log^2 n log r).
- These paths partition the faces of G^* as before, but pi contains at most O(r) vertices per band of the partition.
- Within each piece, use Reif's divide-and-conquer approach with the normal linear time Henzinger et al. algorithm. Doing so takes O(n log r) time total across all bands.
- The total running time is O(n + n log r + n / sqrt{r} log^2 n log r + n log r). Setting r to log^4 n, you get a running time of O(n log log n).