CS 7301.003.20F Lecture 5—August 31, 2020

Main topics are #planar_graphs.

Minimum Spanning Trees

- Last time, we discussed duality in plane graphs and finished up with Euler's formula, including a proof that all simple plane graphs have at most 3n - 6 edges and at most 2n - 4 faces.
- Today, we're going to use that fact.
- A *minimum spanning tree* of a connected edge-weighted graph G is a spanning tree of G with minimum total edge weight.
- We'll assume all edges have distinct weights, which implies the minimum spanning tree is unique. We'll also assume the graph is simple.
- In most algorithms classes, you learn a couple O(m log n) time algorithms for minimum spanning trees in arbitrary graphs. Maybe you learn an O(m + n log n) time algorithm also.
- Even the basic algorithms run in O(n log n) time on a simple planar graph, but today, we're going to do better.
- Tarjan ('83) observed that all the classical minimum spanning tree algorithms follow the same general strategy. Color an edge of G *blue* if it is the lightest edge of some bond or *red* if it is the heaviest edge in some cycle.
- You may have seen a proof of the following lemma or one similar in a past algorithms class.
- Lemma: Let e be any blue edge, and let T be the minimum spanning tree of G / e. Then T cup e is the minimum spanning tree of G.
- Also Lemma: Let e be any red edge. The minimum spanning tree of G \ e is also the minimum spanning tree of G.
- Corollary: Every edge is either red or blue, and the minimum spanning tree is exactly the blue edges.
- So the general strategy is to find one or more blue edges, contract them, and recurse.
- To make fast algorithms from this strategy, we need to get rid of obviously useless edges. An edge is *redundant* if it is a loop or it is not the lightest edge between its endpoints. Every redundant edge is the heaviest edge of a cycle of length 1 or 2, so they are all red.
- We can remove all redundant edges in O(m + n) time using a process called *flattening* of the graph. Now the graph is simple.
    - In short, we do two scans of the adjacency or incident list representation of the graph.
    - In the first scan, we remove all the loops and also essentially build the adjacency list representation of the reversal graph. Because we go over the outgoing darts of each vertex one-by-one, we'll add all darts u ➜ v to the list for v during our scan of u,

guaranteeing these darts appear consecutively in the final list for v.

- In the second scan, we look through each list of incoming edges, for which we just guaranteed parallel edges appear consecutively. We delete from our graph all but the lightest of any set of parallel edges.

- Bor˚uvka (1926) proposed the following algorithm: simultaneously contract the lightest edge incident to every vertex, flatten the contracted graph, and recurse on the resulting minor.

```
BORŮVKA(G):
  if G has no edges
      return ∅
  L ← ∅
  for each vertex v of G
      add the lightest edge incident to v to L
  return L ∪ BORŮVKA(FLATTEN(G / L))
```

Borůvka's minimum spanning tree algorithm.

- We can contract every edge in L in only O(m) time by labeling which component every vertex belongs to, building a new graph with one vertex per component, and then adding darts to the new graph between the old darts' endpoints' vertex labels.
- Each round reduces the number of vertices by at least a factor of 2, so the algorithm performs O(log n) recursive calls and takes O(m log n) time. This analysis works for general graphs.
- But the number of edges in a flattened planar graph with n vertices is only O(n). So the time taken for each iteration is proportional to the number of vertices remaining which is halving every iteration. The time spent per iteration decreases geometrically and the sum is asymptotically bounded by the time taken for the first iteration.
- Theorem: Bor˚uvka's algorithm computes the minimum spanning tree of any simple connected edge-weighted planar graph in O(n) time.
- Again, this algorithm works for arbitrary graphs, but we get the O(n) running time for planar graphs even without knowing an embedding.
- If we're given an embedding, we can do something a bit more simple.
- We use a modification of a strategy by Mares (2004). For the modification, we find a vertex of degree at most 5 using a table bucketing vertices by their degree, contract the vertex's lightest edge, flatten the resulting graph, and recurse.

```
MAREŠEMBEDDED(G):
  T ← ∅
  G ← FLATTEN(G)
  while G has edges
      v ← any vertex of G with degree at most 5
      e ← lightest edge incident to v
      G ← FLATTEN(G / e)
      Add e to T
  return T
```

A minimum spanning tree algorithm for embedded planar graphs.

- Let e be the edge we contract.

- I claim that because G is simple, G / e has at most two pairs of parallel edges. Specifically, if right(e^+) is a triangle, then the edges carrying next(e^+) and next^-1(e^+) are parallel in G / e; symmetrically, if right(e^-) is a triangle, then next(e^-) and next^-1(e^-) are parallel.
- These parallel pairs appear consecutively in the rotation system of G / e so we can find them and flatten them in only O(1) time.
- So we spend constant time each over n - 1 iterations, or O(n) time total.

Straight-Line Embeddings and Schnyder Woods

- Let's discuss a bit more about their actual embeddings in the plane.
- So far, we have not assumed that planar embeddings are in any way well-behaved. For all we know, edges may be embedded as arbitrarily pathological paths.
- However, we can actually show that every planar graph has a *straight-line embedding* in which every edge is embedded as a single line segment.
- Say two planar graph embeddings are *equivalent* if they have the same rotation system.
- Theorem: Every planar embedding of a simple planar graph G is equivalent to a straight-line embedding of G.
- Erickson first describes an inductive proof by de Fraysseix, Pach, and Pollack ('88, '90), but we're going to skip that and go straight to a proof by Schnyder ('89, '90) that further implies the vertices can lie on an n x n integer grid.
- First, if any face of G has degree greater than 3, there must be a path (in the plane) between two non-adjacent vertices. We can add this path as an edge to get a planar embedding of a larger simple plane graph. So we'll assume without loss of generality that G is a triangulation.
- Call a vertex or edge of G *internal* if it is not on the outer face.
- We're going to find a *normal labeling* of G that assigns colors to the corners of each bounded face.
    - The corners of each face are colored red, green, and blue in counterclockwise order.
    - The corners around each internal vertex are colored red, green, and blue in counterclockwise order.

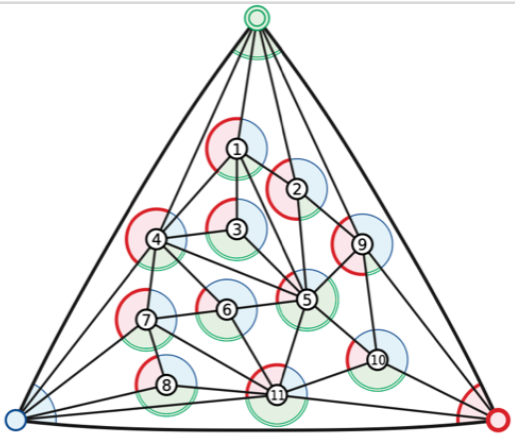

Figure 2.1. A normal labeling of a planar triangulation.

- Finding the labeling takes linear time as so:
- First, color the outer vertices red, green, and blue in counterclockwise order. If G has a single bounded face, its corners inherit the colors of the incident vertices.
- Otherwise, choose an arbitrary edge from a boundary vertex u to an interior vertex v. Vertex v has at least two neighbors in common with u, or the two incident faces for uv would not be triangles.
    1. Suppose v has exactly two neighbors in common with u. The contracted graph G / uv has two pairs of parallel edges; each pair was a triangle of G. We delete one edge from each pair to get a smaller triangulation H.
        - Then, we recursively compute a normal labeling for H and transfer corner colors back to G.
        - For the faces on either side of uv, we have the corners incident to the boundary vertex u inherit u's color and then label the other corners as required.

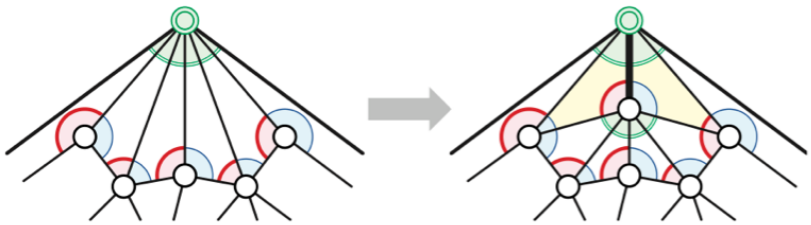

Figure 2.2. Recursively computing a normal labeling.

    2. If v has at least three common neighbors with u, then there is a triangle uv with at least one such vertex in its interior. We recursively compute normal labelings of the subgraphs of G inside uvw and outside uvw. Boundary vertex u retains its original color in both subproblems, so the resulting labelings will be compatible.
        - But if we think a bit deeper, we realize case 2. really just leads to us to eventually finding an edge for case 1. We can even number the vertices in the order in which they get contracted by uses of rule 1.
- Every internal edge in G is incident to corners with all three colors, with one color appearing twice at one endpoint. Orient each internal edge toward the endpoint with the

same color twice, and assign the repeated color to the edge. This coloring and orientation is called a *Schnyder wood*.

- Each boundary vertex has only incoming internal edges, all of the same color.
- Each internal vertex has exactly one outgoing edge of each color.
- At every internal vertex we see in counterclockwise order: one outgoing red, 0 or more incoming blue, one outgoing green, 0 or more incoming red, one outgoing blue, 0 or more incoming green.

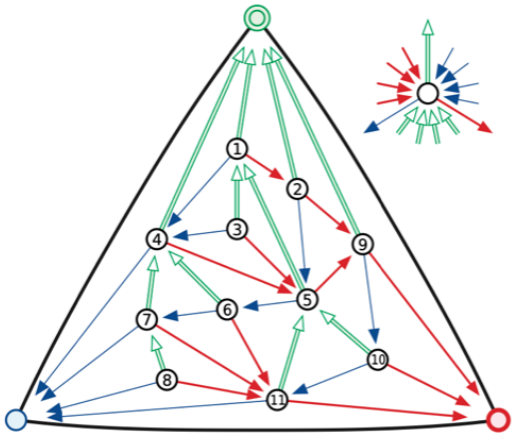

Figure 2.3. The Schnyder wood constructed from the normal labeling in Figure 2.1.

- Lemma: In any Schnyder wood, the edges of each color induce a spanning tree of the internal vertices, rooted at the boundary vertex of that color.
  - We'll assume we always contracted into the green boundary vertex.
  - Number the vertices by the order in which they are contracted to the green boundary vertex. Label the green boundary vertex 0 and the other two boundary vertices infty.
  - Every interior vertex has exactly one outgoing edge of each color. The green edges have strictly decreasing labels, but the blue and red edges have strictly increasing labels.
- We're finally about to assign integer coordinates for the vertices' embedding.
- Each internal vertex v has a unique directed path of red edges, a unique path of blue edges, and a unique path of green edges. Together with the outer face edges, these paths partition the triangulation into three regions. Each region is given the color not used by its two colored bounding paths. We say each region includes every vertex in its interior and its counterclockwise bounding path, but not v itself.

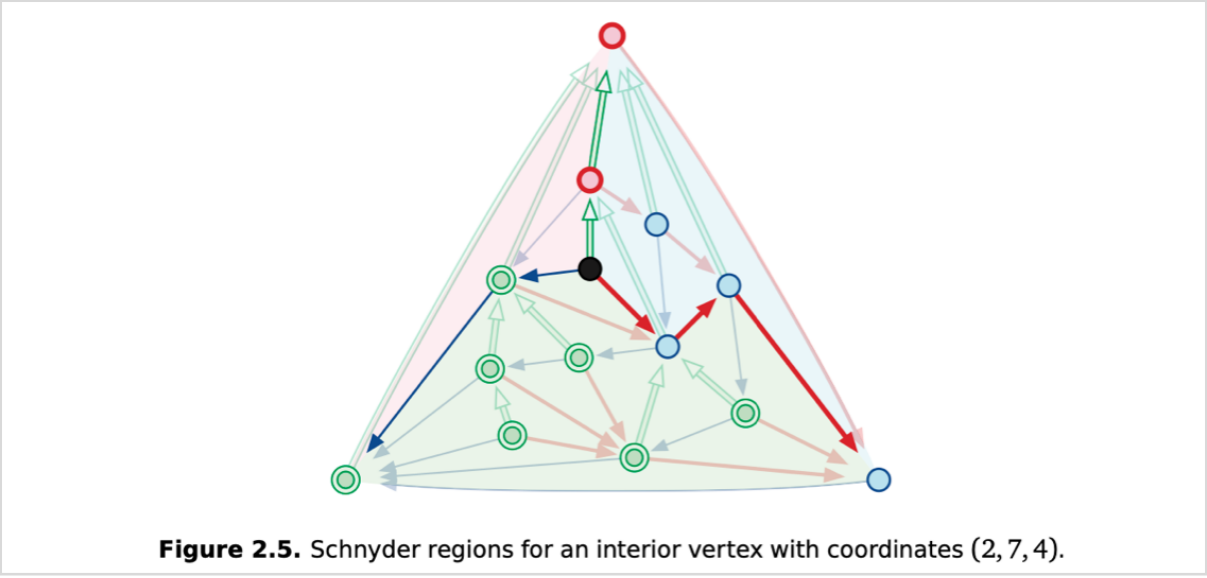

Figure 2.5. Schnyder regions for an interior vertex with coordinates $(2, 7, 4)$.

- For any interior vertex v, let r(v), g(v), and b(v) respectively denote the number of vertices in the red, green, and blue regions of v.
  - Also, the red boundary vertex has r(v) = n - 1, g(v) = 0, and b(v) = 1.
  - The green boundary vertex v has r(v) = 1, g(v) = n - 2, and b(v) = 0.
  - The blue boundary vertex has r(v) = 0, g(v) = 1, and b(v) = n - 2.
- We have r(v) + g(v) + b(v) = n - 1 in each case.
- Lemma:
  - For red edge v ➜ w, we have r(v) < r(w) and g(v) ≥ g(w), and b(v) > b(w).
  - For green edge v ➜ w, we have r(v) > r(w) and g(v) < g(w), and b(v) ≥ b(w).
  - For blue edge v ➜ w, we have r(v) ≥ r(w) and g(v) > g(w) and b(v) < b(w).
  - For every triangle uvw with corners colored red, green, and blue by the normal labeling, r(u) ≥ r(v) > r(w) and g(v) ≥ g(w) > g(u) and b(w) ≥ b(u) > b(v)
- Proof:
  - Consider a green edge v ➜ w and find the regions for v. As we change to the regions for w,
    - v changes from uncolored to green
    - w changes from red to uncolored
    - every green vertex remains green
    - no vertex changes from red to blue or vice versa
  - Cases for red and blue edges are symmetric. The triangle claim follows from just analyzing the cases.
- Theorem: Any planar embedding of a simple planar graph with n vertices is equivalent to a straight-line embedding with vertices on the (n - 1) x (n - 1) integer grid.
  - Assign each vertex v the integer coordinate (g(v), b(v)).
  - For any triangle uvw with red, green, and blue vertices, its orientation is determined by the determinant

$$\begin{vmatrix} 1 & g(u) & b(u) \\ 1 & g(v) & b(v) \\ 1 & g(w) & b(w) \end{vmatrix} = \big(g(v)-g(u)\big)\big(b(w)-b(u)\big)-\big(g(w)-g(u)\big)\big(b(v)-b(u)\big).$$

which is positive from the previous lemma.

- So every triangle is oriented counterclockwise, which implies no overlapping triangles or edges.
- In particular, the signed area of the outer triangle is equal to the signed areas of the interior triangles. There cannot be overlaps, because otherwise the signed area would be larger than the signed area of the outer triangle.

- We could have also assigned each vertex a coordinate ((r(v), g(v), b(v)) to embed in the triangular grid x + y + z = n - 1.