Math 3322: Graph Theory[1] Mikhail Lavrov

Lecture 17: Eulerian tours

October 14, 2021 *Kennesaw State University*

1 Two problems

In this lecture and the next, we will consider two very similar-sounding problems.

1. An **Eulerian tour** in a graph G is a closed walk that uses every edge of G exactly once.
2. A **Hamiltonian cycle** in a graph G is a cycle[2] that visits every vertex of G exactly once before ending up back where it started.

We call a graph Eulerian if it has an Eulerian tour, and we call a graph Hamiltonian if it has a Hamiltonian cycle.

It will turn out that it's easy to determine if a graph is Eulerian. On the other hand, figuring out if a graph is Hamiltonian or not is hard. We will see a few situations in which we can answer that question, but a lot of the time, we will not know the answer except by brute force.

1.1 Why line graphs don't help

In the previous lecture, we saw that line graphs help us turn edge problems into vertex problems. So how can it be that one of these problems is easy, and one is hard?

It turns out that a kind of connection between the two problems exists...

Claim 1.1. *If G is Eulerian, then $L(G)$ is Hamiltonian.*

Proof. If we take an Eulerian tour $(v_1, v_2, v_3, \ldots, v_m, v_1)$ of G, then the sequence

$$(v_1v_2, v_2v_3, \ldots, v_mv_1, v_1v_2)$$

is a Hamiltonian cycle in $L(G)$. □

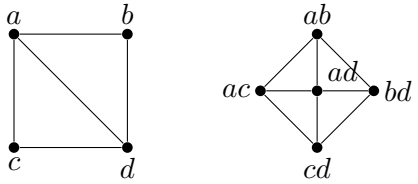
...but there are two problems that mean this connection is not particularly strong.

Problem #1: A path or walk in $L(G)$ does not always give us a path or walk in G, and in particular a Hamiltonian cycle in G does not always give an Eulerian tour in $L(G)$.

We've already seen the following example of a graph and its line graph:

[1] This document comes from the Math 3322 course webpage: http://facultyweb.kennesaw.edu/mlavrov/courses/3322-fall-2021.php

[2] we could have said "closed walk" to match the previous problem, but if your closed walk does not revisit vertices, then it's a cycle.



The graph on the right has a Hamiltonian cycle: for example, (ac, ad, ab, bd, cd, ac). But when we try to combine these edges into a closed walk in the graph on the left, it doesn't work: we can't take edge ac, then edge ad, then edge ab. (The line graph makes sure that these edges share an endpoint, but it doesn't check that they share the correct endpoint.)

In the previous application, this wasn't a problem; a walk in $L(G)$ can be turned into a walk in G if you're also allowed to skip some of the vertices of the walk in $L(G)$. But if you do that here, and skip edge ad, you won't have an Eulerian tour anymore.

Problem #2: Not all graphs are line graphs, so the reduction only goes one way.

Even if there were a perfect relationship between Eulerian tours in G and Hamiltonian cycles in $L(G)$, it wouldn't help us find a Hamiltonian cycle a graph H that's not the line graph of any graph. In the previous lecture, we proved the vertex version of Menger's theorem, and then used it to prove the edge version; the reverse wouldn't work.

2 Finding Eulerian tours

The following result has a very short proof:

Lemma 2.1. *If a graph G has an Eulerian tour, then every vertex in G has an even degree.*

Proof. Every time the Eulerian tour enters a vertex v, it must also leave, using two edges out of v. If, after doing this k times, all the edges out of v are used up, then v must have degree $2k$, which is even. □

Another way to say this: the condition "every vertex in G has a even degree" is **necessary** for the graph to be Eulerian. No graph can be Eulerian without this condition.

We would also like to know if this condition is **sufficient** for the graph to be Eulerian: is checking this condition enough to know that G has an Eulerian tour?

Thinking about necessary and sufficient conditions is the key to understanding the difference between the Eulerian tour problem and the Hamiltonian cycle problem.

- For the Eulerian tour problem, we will be able to state a condition which is both necessary and sufficient (and which can be checked without doing very much work).
- For the Hamiltonian cycle problem, we will find some necessary conditions, and we will find some sufficient conditions, but as far as we know there is no condition that is necessary, sufficient, and easy to check.

(It will be important in the next lecture to keep track of *which* conditions are necessary conditions, and which ones are sufficient conditions!)

Back in the world of Eulerian graphs, before we prove the big result we want, we will take a detour[3] to discuss cycle decompositions.

2.1 Cycle decompositions

In general, a "decomposition" of a graph just means separating the graph into many pieces that share no edges. For instance, we say "block decomposition" because every edge is in a block, and two blocks never have any edges in common. (We usually don't care what the vertices do in a decomposition; in particular, isolated vertices can always be ignored.)

A **cycle decomposition** of G, then, is a set of cycles in G such that every edge of G is in exactly one cycle.

Lemma 2.2. *A graph G has a cycle decomposition if and only if every vertex of G has an even degree.*

Proof. The "only if" direction is due to the fact that every cycle contributes 2 to the degree of the vertices it passes through. So if G has a cycle decomposition, then the degree of a vertex that lies on k of the cycles is $2k$, which is even.

To prove the "if" direction, we will use strong induction on the number of edges in G. As our base case: if G has 0 edges, then the empty set is a cycle decomposition: it includes every edge exactly once!

Suppose G has $m > 0$ edges. Pick any component of G that has an edge in it. That component has minimum degree 2 (because all the degrees in it are even and positive), so we know it contains a cycle C.

In $G - C$ (the graph we get by deleting the edges of C, leaving the vertices as they are), all the degrees are still even: the degrees of vertices of C went down by 2, and the rest are unchanged. Also, $G - C$ has fewer than m edges (at most $m - 3$), so by induction, $G - C$ has a cycle decomposition.

Add C to that cycle decomposition, and we get a cycle decomposition of G; by induction, this exist for any number of vertices. □

This proof also suggests an algorithm for finding a cycle decomposition. Simply find any cycle (which we can do by aimless walking around until we revisit a vertex), set it aside, and repeat with the remainder of the graph until we are out of edges.

2.2 Gluing together cycles

The cycle decomposition lemma lets us prove the big result we actually wanted:

Theorem 2.3. *A graph G is Eulerian if and only if it is connected (except possibly for isolated vertices) and every vertex of G has an even degree.*

[3] Heh-heh.

Proof. We know that the even-degree condition is necessary by Lemma 2.1. We also know that an Eulerian graph cannot have two edges in different connected components: in that case, a tour would never be able to visit them both. So it remains to show that together, these two conditions are sufficient.

Assume that G has only one connected component that is not an isolated vertex, and every vertex of G has an even degree. The first thing we'll do is use Lemma 2.2 to find a cycle decomposition of G.

Now, we'll build up our Eulerian tour step by step. This is a proof by algorithm. Throughout this process, we will always have two things:

- a "partial tour" PT, which is a closed walk that walks each edge at most once, but might miss some edges.
- a set $\mathcal{U}$ of "unprocessed cycles": a cycle decomposition of just the edges that are **not** part of PT.

Initially, we'll take PT to be one of the cycles in our cycle decomposition, and $\mathcal{U}$ to be all the other cycles.

Then, one by one, we'll remove a cycle from $\mathcal{U}$, and splice it into PT. To make this possible, it's important to find a cycle in $\mathcal{U}$ that contains one of the vertices visited by PT. There must always be such a cycle: otherwise, we'd get two connected components in G! There would be no edge between two disjoint sets: the set of vertices visited by PT, and the set of vertices that are part of a cycle in $\mathcal{U}$. Both of these sets contain some edges. That would contradict our assumption about G.

So now suppose we've found such a cycle C. Say that $PT = (w_0, w_1, \ldots, w_t, w_0)$, $C = (v_0, v_1, \ldots, v_k, v_0)$, and these share vertex $w_i = v_j$. Then replace PT by the closed walk

$$(w_0, w_1, \ldots, w_{i-1}, w_i = v_j, v_{j+1}, \ldots, v_k, v_0, \ldots, v_{j-1}, v_j = w_i, w_{i+1}, \ldots, w_t, w_0).$$

Essentially, insert C into PT at a point where the shared vertex is visited.

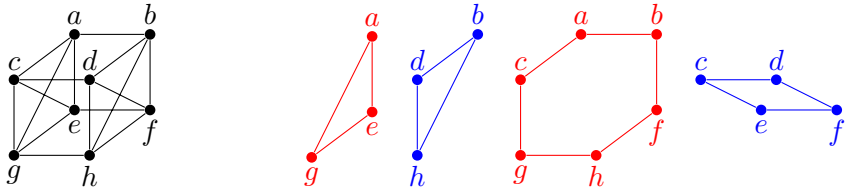
Note that after this process, PT walks all the edges which were previously traversed by C. Therefore after we do this (and remove C from $\mathcal{U}$), it's still true that $\mathcal{U}$ is a cycle decomposition of the edges not part of PT. Also, it's still true that PT uses every edge at most once: none of the new edges were previously used by PT.

Repeat this until $\mathcal{U}$ is empty; then PT is an Eulerian tour. □

The algorithm we get here is more or less the algorithm known as **Hierholzer's algorithm** for finding Eulerian tours.

2.3 An example

Let's use this method to find an Eulerian tour in the following modified version of a cube graph (on the left):



Our first step is to find a cycle decomposition. To do this, just take any cycle in the graph, remove it, and keep going with the rest. One possible decomposition is into the four cycles on the right.

(It is not the shortest one; there's actually a decomposition into two cycles of length 8. If we wanted to find the Eulerian tour as quickly as possible, the shorter decomposition would be better. But here, I want to demonstrate the method in action.)

Now let's splice them together into a tour.

1. Initially, our partial tour PT is just the first cycle: (a, g, e, a).
2. Next, we want to add a cycle that shares a vertex with PT. We can choose either the third or the fourth cycle (the second one doesn't work). Let's take the fourth cycle: (c, d, f, e, c).

   The only vertex this shares with PT is e. So we follow PT from a to e, then walk around the cycle starting at e and returning to e, then follow the remainder of PT. This gives us:

   $$(a, g, e, c, d, f, e, a).$$

3. At this point, PT has lots of vertices, so we can choose any cycle to add. Let's add the second cycle: (b, d, h, b). We can splice this into PT at vertex d, getting:

   $$(a, g, e, c, d, h, b, d, f, e, a).$$

4. Finally, we add the only remaining cycle: (a, b, f, h, g, c, a). Since PT and this cycle both start and end at a, combining them is particularly easy. Just follow one then the other, getting

   $$(a, b, f, h, g, c, a, g, e, c, d, h, b, d, f, e, a).$$

The result is an Eulerian tour; you can check this by tracing it out, and seeing that you get all the edges of the original graph G without doubling back over any edge.