Cycle Bases

Mary Radcliffe

The goal of this document is to understand what a cycle basis is, and how to construct one. In order to do this, we'll first start with some preliminaries about cycles and subgraphs, and then build the necessary structures.

1 Preliminaries

We have seen graphs used as examples in class, and constructed adjacency matrices from these graphs. Here, we'll need to use some special kinds of graphs. Specifically:

Definition 1. *A cycle in a graph is a list of vertices $v_1, v_2, \ldots, v_n, v_1$ such that v_i is adjacent to v_{i+1} for all $i \leq n-1$, v_n is adjacent to v_1, and no vertices are repeated.*

That definition is pretty formal looking, but it basically describes a logical thing: just a walk around a bunch of vertices, no backtracking. See, for example, Figure 1.

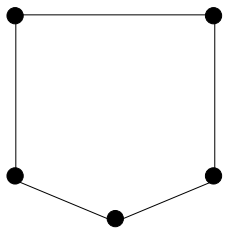


Figure 1: A cycle of length 5 (meaning having 5 vertices)

One note to make here is that while we sort of defined a "starting place" to define the cycle, we usually don't think of cycles having a designated "first" vertex. That is to say, the cycle v_1, v_2, v_3, v_1 is the same as the cycle v_2, v_3, v_1, v_2. These two cycles denote the same set of vertices, and in the same order, and only differ in which one came first on the list.

In general, here, we will be interested in the set of all cycles a graph can have. See, for example, Figure 2. In this figure, we have a graph G, and we see that there are 3 different cycles in G.

Specifically, these cycles in red, green, and blue are part of G, and we refer to them as subgraphs. Very specifically, we have the following definition:

Definition 2. *A subgraph H of a graph G is a collection of some vertices from G and some edges from G. That is, if u and v are adjacent in H, they must also be adjacent in G, but not vice versa.*

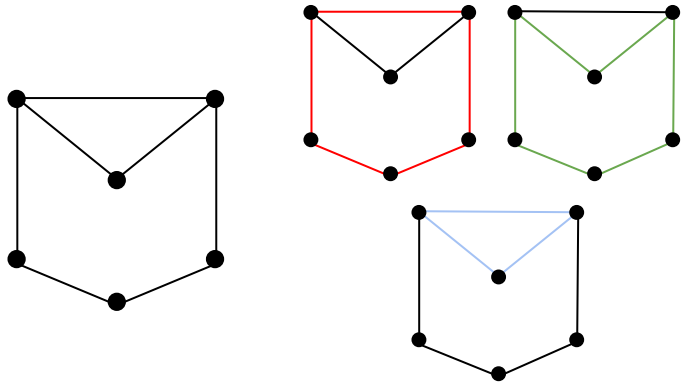


Figure 2: On the left, we see a graph G. On the right, outlined in different colors, are all the cycles in G, also called the cycle set of G.

This is all a fancy way of saying that a subgraph of a graph G is any graph you can build by erasing some of the vertices and/or edges in G, but not adding any new ones. Importantly, if you erase a vertex, you also have to erase the edges that touch that vertex.

Definition 3. *A connected graph without any cycles is called a tree.*

Trees show up throughout graph theory as sort of the backbone of any graph. Indeed, every graph has a subgraph that includes every vertex and is a tree. Such a subgraph is called a *spanning tree*.

Definition 4. *A spanning tree in a graph G is a subgraph of G that includes every vertex of G and is also a tree.*

We note here that there are often lots of different spanning trees for a given graph. See Figure 3 for some examples.

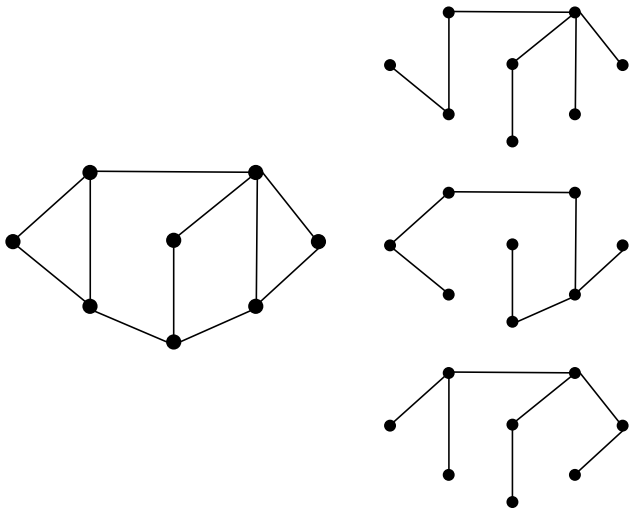
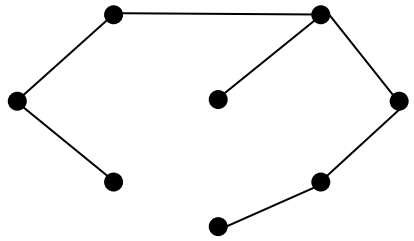


Figure 3: On the left, we see a graph G. On the right, we see three different spanning trees of G. All these trees contain every vertex of G, but no two are the same. There are other spanning trees possible here, as well.

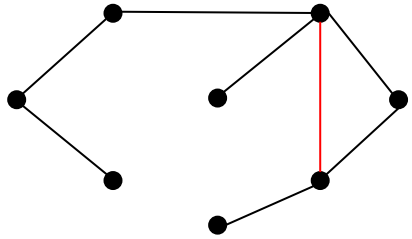
To construct a spanning tree for a given graph is simple. First, start with the graph. Then, iteratively remove any edge that appears in a cycle. Once all cycles are destroyed, the remaining graph is a tree.

Indeed, thinking about this process backwards (starting with a spanning tree, then iteratively adding edges) will be integral to our construction of a cycle basis for G. Indeed, let's consider what happens when we add edges, one by one, to a spanning tree.

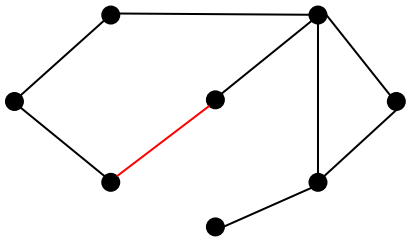
We start with the following tree:



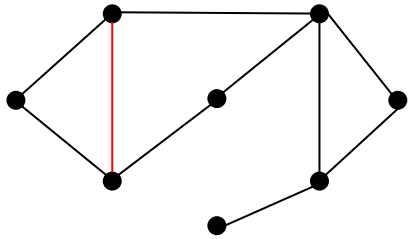
Then we'll add one edge, shown in red below. Notice that by adding an edge to a tree, we're forced to add a cycle. In particular, we add exactly one new cycle, whose edges are all from the tree except the one red edge.



Now, we'll do this again, and notice the same thing occurring. We add exactly one edge, which makes exactly one new cycle, using the edge we made and edges from the tree.



So, is this always what's going on? Alas, no, as the last red addition shows. We here add one new edge, but *two* new cycles appear. In this case, one of the new cycles includes our new red edge plus only edges from the original tree, but the other includes the edge added at the second step.



However, we do have the following useful Lemma, whose proof I shall leave as an exercise.

Lemma 1. *Let G be a graph, and let T be a spanning tree of G. For every edge e not in the spanning tree, there is a unique cycle that includes the edge e and has all remaining edges in T.*

That is to say, when we add edges one by one, even if we create more than one cycle by doing this, we only create ONE cycle that has all its other edges present in the spanning tree. The proof of this is straightforward by contradiction: if there were more than one such cycle, it would necessarily mean there is a cycle in T, which, by definition, is impossible.

Our finally item of note for this preliminaries section is a definition of an Eulerian graph. Fundamentally, we can think of an Eulerian graph as a graph that is built entirely out of cycles. Consider the following example:

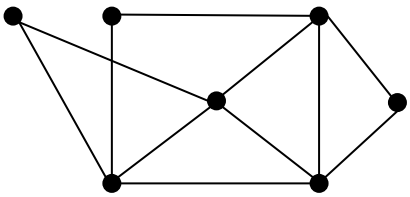


Figure 4

Notice that we can break this graph up into a collection of cycles.

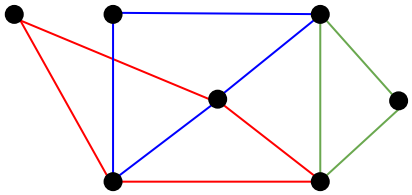


Figure 5

One important observation about Eulerian graphs is that the number of edges at each vertex (this is called the degree of the vertex) is always even. This is clear to see based on how we have built the graph. Since the graph is made

up of a bunch of cycles, each vertex has an option: to be in each cycle, or not. If it's in a particular cycle (for example, in the red cycle), that cycle contributes two edges to its degree (that is, two red edges touch the vertex, for example). If not, it contributes 0 edges. Hence every vertex has evenly many edges incident to it.

I'll note here that this is not the typical definition of an Eulerian graph. The typical definition has to do with walks, and is pretty interesting in its own right. If you want to know a little history about graph theory, and where this idea came from, you should Google the Bridges of Königsburg problem, widely credited as being the Problem That Started Graph Theory.

Our final note on Eulerian graphs is that the decomposition into cycles isn't unique in any way. Here's another way to do it for the graph above, for example.

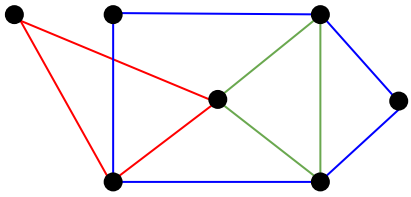


Figure 6

In this example, we have the same number of cycles as in the first decomposition, but that's sheer coincidence. It may not be that way, and that's ok.

2 A moment for practicality

So, cool, we have trees, and cycles, and Eulerian graphs, and, uh, who cares?

Well. Imagine, if you will, that you might be building a circuit, or electrical network. In this case, your vertices are nodes of the network, and edges are representing wires between nodes. Often in this case, the edges are directed (indicating the direction that voltage will be carried) and weighted, as different wires have different carrying capacities.

What is important to you, when constructing this network, is that voltage should balance, so that you aren't accumulating excess energy at a given point. This is essentially Kirchhof's Second Law of Voltage: around any closed loop in a network, the sum of voltage drops should be 0.

In the next section, we will show how to construct a vector space over the Eulerian subgraphs of a graph G, called the *cycle space* of G, and how to build a basis to generate that vector space. And here, my friends, is the really cool thing.

Theorem 1. *Let G be a (weighted, directed) graph, and let V be its cycle space. Then G satisfies Kirchhof's Second Law if and only if G satisfies Kirchhof's Second Law on a basis for V.*

That is to say, if we can understand how to construct a basis for the cycle space, we can verify Kirchhof's Laws JUST by looking at the basis, and not having to construct every single closed loop in the network.

I am not going to prove this theorem here, though I will touch on it briefly

in a later lecture. But hopefully you are sufficiently convinced that this is not in fact an exercise in futility. Or, perhaps, you disagree, but we shall continue on our futile project nonetheless.

3 A vector space and a basis

Let G be a graph, and let V be the set of Eulerian subgraphs of G. We define a vector space on V, called the cycle space of G, over $\mathbb{Z}_2$ using the following addition:

> Given $H_1, H_2 \in V$, we define $H_1 \oplus H_2$ to be the subgraph of G that contains all edges that appear in exactly one of H_1 or H_2.

Another way to think of this sum is to think that we union up all the edges in H_1 and H_2, but since we are working over $\mathbb{Z}_2$, any edge that appears twice gets annihilated. An example of this perspective is shown in Figure 7.

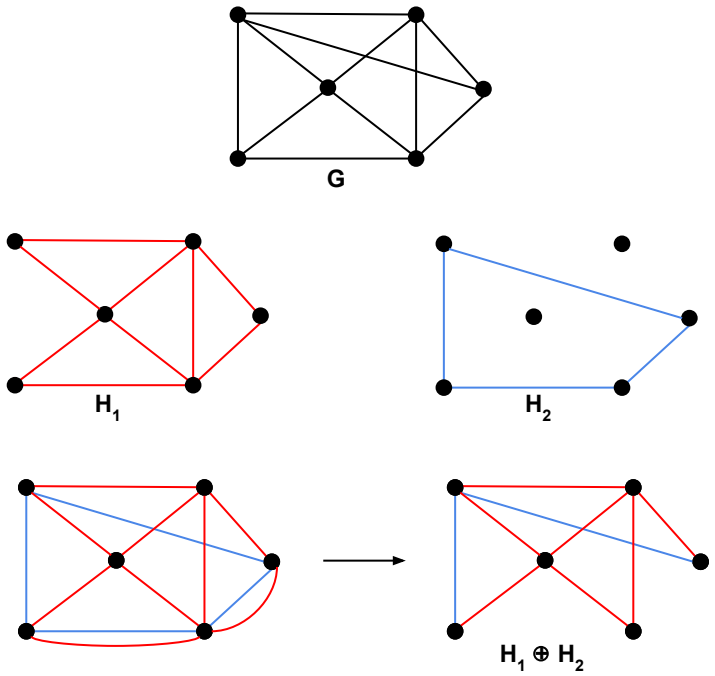


Figure 7

Note that since we are working over $\mathbb{Z}_2$, we don't need to define scalar multiplication, because there are only two possible scalar products: multiplying by 0, which yields the empty graph, or multiplying by 1, which yields the same subgraph back.

In order to show that this is a vector space, we would need to demonstrate that this definition of addition and scalar multiplication satisfies the 10 conditions we have seen before. Most of these are fairly trivial; perhaps of interest is only the closure under addition. Note that if we simply added the edges without cancellation of duplicates, this is trivial; if every vertex has even degree in each of H_1 and H_2, then the sum of the two subgraphs clearly has even degree. If an edge is duplicated, note that we subtract 2 from the sum of the two subgraphs:

one from H_1 and one from H_2, so parity is maintained at each vertex (you can think of this as adding red degree + blue degree, and then if we subtract an edge we subtract one from red and one from blue). I leave the remaining properties as an exercise.

A note here: this writing of the vector space is a bit clunky. It's cumbersome to have to draw out a graph every time you want to refer to a vector in the space. So we can identify this with a subspace of $\mathbb{Z}_2^m$, where m is the number of edges in G, as follows: number the edges in G from 1 to m. Given a subgraph H of G, we identify the subgraph with the vector that has a 1 in the i position if edge i is in H, and 0s elsewhere. Using this presentation of the graphs/vectors in H, the addition we have defined above is just traditional vector addition in $\mathbb{Z}_2^m$, so V will in fact be a subspace of $\mathbb{Z}_2^m$.

Now, let us think about how we can build a generating set for this vector space. From the definition of Eulerian graphs as just the union of a bunch of cycles, certainly we could form a generating set just by taking all the cycles in G. Then, every Eulerian graph can be built by adding up appropriate cycles.

However, as we have seen in the example in Figures 4, 5, and 6, this generating set is not linearly independent. Indeed, in this example, we have an Eulerian graph that has two different representations as a sum of cycles. By definition of linear dependence, then, the set of cycles in G is, well, not. So we'd like to be able to construct a generating set for the space that IS linearly independent.

Here's where that stuff about trees from the Preliminaries comes back. Recall that given a graph, we can look at a spanning tree of that graph, and then build the graph back up bit by bit by adding one edge at a time. Recall also that when we do that, we add in cycles bit by bit, and by Lemma 1, for each edge we add in *exactly* one cycle that has all its other edges in the original T.

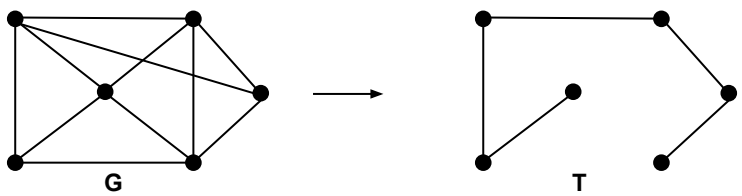
This leads us to a super amazing cool theorem.

Theorem 2. *Let G be a graph, and let V be its cycle space. Let T be a spanning tree of G, and for each edge e not in T, let C_e be the unique cycle in G that contains e and has all other edges in T. Then $\{C_e \mid e \notin T\}$ is a basis for V.*

Recall, from our interlude into practicality, that if we are considering an electrical network and wish to confirm Kirchhof's Second Law of Voltage, we need only check the Law on a basis for V. So, we could just pick a spanning tree, construct the basis as described in the theorem, and then quickly check that Kirchhof's Law is satisfied. This saves us a whole bunch of time if there are lots of cycles in the graph, since only a relatively small number of them must be checked.

We close this set of notes with an example of constructing a cycle basis for a graph.

First, we start with a graph G, and identify a spanning tree T in G:



Then, we look at $T \oplus \{e\}$, for every edge e not in the tree T. Each such graph (there are 6 of them) contains exactly one cycle. We isolate that cycle and add it to our cycle basis.

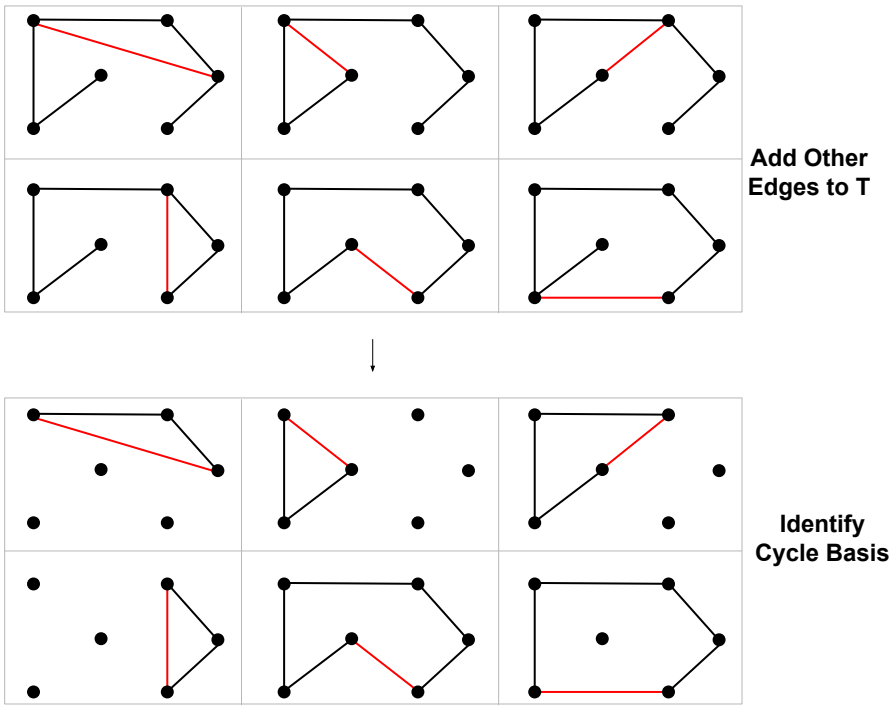
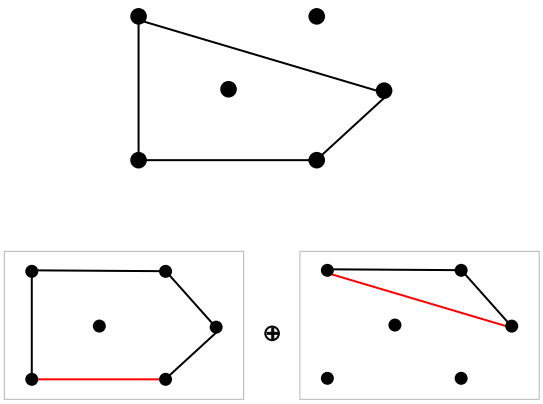
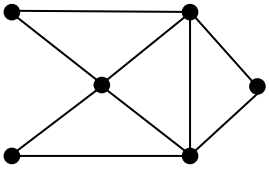


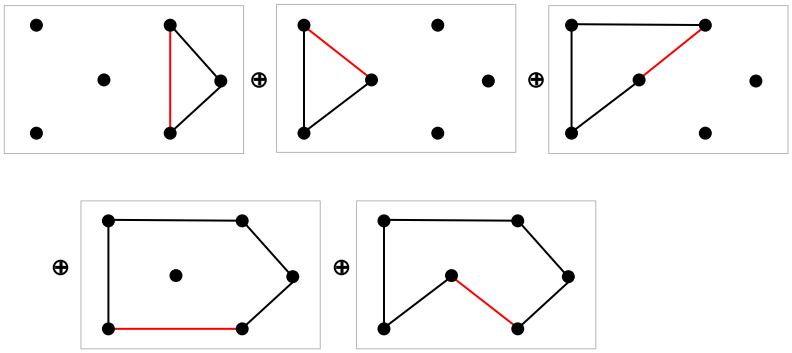
Figure 8

From here, any Eulerian subgraph of G (that is, any element of the cycle space) can be written as a $\mathbb{Z}_2$-linear combination of these cycles. For example, let's take the two Eulerian subgraphs we used in Figure 7. Here is H_2 written in terms of the cycle basis found in Figure 8.



And here is H_1:





Math rules.