ERIK DEMAINE: All right. Today we start a new section of NP-hardness. I think it'll be one lecture about Hamiltonicity. So Hamiltonian cycle and path.

We've talked about this briefly. But cycle is just a cycle visiting every vertex exactly once. And usually when you talk about a graph being Hamiltonian, you mean that there's a Hamiltonian cycle, also called a Hamiltonian tour, or Hamiltonian circuit all mean the same thing.

And then there's Hamiltonian path. Another problem we have looked at before. And same thing, but path that visits every vertex exactly once. These are pretty much the same problem, but different versions are useful in different settings.

In case you're curious where this comes from, it comes from this game introduced by Hamilton in 1857. He was the royal astronomer of Ireland back then, but he had fun with games as well.

So here you have something like the graph of a dodecahedron. The dual is an icosahedron, hence the name, and you want to find a Hamiltonian circuit through it. Or there's actually a two-player game where one person chooses a subset of the graph, the other one tries to find Hamiltonian circuit in that subset.

So that's where Hamiltonicity comes from. But for us today it is a nice NP-complete problem. We've actually seen one proof that planar max degree four, I think, Hamiltonicity is hard. That was in the planar SAT discussion, and this is the original paper that does planar 3SAT.

And there was this proof-- I guess it's not actually max degree four, probably a max degree six or something. We argued the directed case-- this apparently works for

the undirected case-- we'll get an even stronger result here.

In general, the problem is NP-complete in a lot of different special cases, such as the one we saw. For example, if you're looking for a path and you're given the two end points-- so normally you'd say is there a Hamiltonian path or is there Hamiltonian cycle?

Those are both hard in general graphs. But if I give you a graph and I give you two nodes, s and t, and I want to know is there-- so here's the graph, here's s and t-- I want to know is there a path that starts at s, ends at t. This is NP-hard. By reduction from Hamiltonian cycle, tell me what to do.

AUDIENCE: You add a new node which has connection to s and t and nothing else.

ERIK DEMAINE: Add a new node connected to s and t and nothing else. Yeah. So we'll add this new node x, connected to s and t.

What's interesting about adding a degree two node is you know it has to be in the cycle, so you know that these two edges have to be in the cycle.

AUDIENCE: Also, if it's directed you can direct those two so that--

ERIK DEMAINE: It's not directed. All these are going to be undirected. This would preserve directedness if you wanted it.

So there you go. So if Hamiltonian cycle's hard, then Hamiltonian path given two end points is hard. Easy warm-up.

It's also NP-complete for planar three regular, three connected graphs, where every face has degree at least five. But probably already met them. It's NP-hard for bipartite graphs. These are all various results proved in the '70s. It's going to be hard for squares of graphs.

Squares of graphs is interesting. This is whenever you have a length two path, you add an edge. It's a square, and that's still hard. Another fun problem is computing a Hamiltonian cycle given a Hamiltonian path is also hard.

On the other hand, some variations are easy. So you have to be a little bit careful. There are a lot of papers about various graphs that are always Hamiltonian and usually when they're always Hamiltonian there's a good algorithm to find them. Here are just a couple of examples.

One is cubes of graphs. So if you take all length three paths and connect them by an edge, or maybe length less than or equal to three-- I'm not sure exactly-- that's easy. All such graphs are Hamiltonian, where with squares it's still hard. So that's kind of a limit.

And the other one here is planar-- three connector graphs is NP-complete. Planar four connector graphs is polynomial. This is a tough theorem, one of many tough theorems. All such graphs are Hamiltonian.

So those are a few cases to worry about. We will see some more today. And these are relatively simple. We're going to prove even stricter versions of this problem NP-complete, but it starts to give you the flavor of what's there.

In fact the first one we will prove is-- this may seem a little weird because I wanted to do undirected, but we're going to make it undirected in a moment. We're going to do a series of reductions.

First one we'll do is planar max degree three graphs. So this will be a reduction from 3SAT. So the general idea, and please wait why you should care about this problem. We're going to reduce it to another more interesting version of Hamiltonicity.

But here's one way to represent 3SAT as Hamiltonicity-- another way, similar to the one we did before. We got clauses here over on the left. Here some of the clauses have size two, but it works for the three case.

Here we have the variables. And the key concept here is this notation. This is shorthand for a more complicated gadget, which is to say that these two things should not be both chosen. This is an exclusive OR. Exactly one of them should be

chosen.

So that forces x1 to be the opposite of x1 bar, and it also lets you duplicate things. So here we get multiple copies of the variable. You keep doing that several times you get many copies of variable and its complement.

And then we're also going to use some connections like this. We're going to need a crossover gadget, but this will say that you-- well, so this is going to be some clause which it's not literally as drawn. Again, this is notation for a gadget the is going to force at least one of these to be traversable.

And so that overall should represent 3SAT. We need a clause gadget. We need this exclusive OR so that we can split, duplicate things. Overall, the cycle's going to do this. It's going to visit all the variables in order, then all the clauses in order. So we can't get anything interesting out of planarity, because that graph with these connections is not going to be planar. So we'll need a crossover. And those are the three gadgets we need.

Yeah.

AUDIENCE: Why were you using exclusively working with [INAUDIBLE] variables [INAUDIBLE]?

ERIK DEMAINE: So the idea is here there's a choice. Do I choose this edge or this edge in the Hamiltonian cycle. And I want the choice for this copy of x1 to be the same as for that copy of x1. So the exclusive OR forces each instance of x1 to be the same. In general, if there's k instances, you'd have k of these.

AUDIENCE: Can you connect that? Use the same guy to connect it to the instance inside the clause?

ERIK DEMAINE: I guess the reason why we need two copies here is that you can't-- well, see you can't have 2 XOR gadgets going to the same place. So it will be clear once we've seen the XOR gadget.

We want two distinct copies so that these two things can point to different copies of x1. That's why we're making a copy like that. Does that make sense?

AUDIENCE: [INAUDIBLE].

ERIK DEMAINE: OK. Excellent. Actually, same slide. I should just show them.

So here is-- maybe let's start with the XOR gadget because the clause-- ah, clause doesn't actually use it.

So the XOR gadget is if we have two edges of opposite direction relative to the way we're being connected. I think it also works in the other way because these are-- no, these are also opposite if you redraw it this way.

We're just going to have this picture, which can be traversed. Either this guy can come in-- here we're really using direction, directionality of the graph. You can traverse the top edge or you can traverse the bottom edge in the opposite direction.

But you can't just turn around because then these vertices would be uncovered. And as soon as you've gone this far, you can't come back this way.

So once you enter here you're forced to leave there. Once you enter here you're forced to leave there, if you want to cover everything. So that's an exclusive OR, and at least one of those better happen.

So that's what happens over there. And also what we'll end up doing here, except we also need the crossing case.

On the other hand, for the clause, the idea is basically there's a cycle over here, which is fine. But somehow that cycle for that clause has to be connected into the big cycle that we've drawn already. And at least one of the variables needs to connect to that.

So the idea is we can come up here-- so also these vertices need to be covered. So we can come up here, go over here, loop around, and be happy. And then we've grab the entire cycle.

Also, when we've grabbed this cycle, we can also grab these vertices. All of them.

That's only going to be possible if this is possible. So in the outer side, we can't go here and then up here. We can go here over here. That's also fine. OK. Good. I see.

So that the issue is-- so these are actually flexible. These guys can be covered basically from the right side or from the left side, either way. So it doesn't matter whether you take this path or this path.

But at least one of these sort of pairs of edges, as they're drawn over here, you need to be able to choose the left choice and, therefore, get the big cycle. And the issue with that is because you have this XOR, it's an exclusive OR, exactly one of those two things must be true.

So in some cases, you'll be forced to choose the right thing. If this variable chose the right path, then this one will also have to choose the right path because of this exclusive OR, and preventing you from grabbing the cycle. You're fine in this clause if at least one of them you choose the left path.

In fact, if you look very closely there's some bold edges indicating which one's got chosen here. So here, two of them were on the right, one of them was on the left for this clause. In general, if at least one of them is on the left you're OK for the clause. Clear.

The high level now. This would work fine if we didn't care about planarity, but I want planar directed max degree three. At this point we should be directed max degree three. So that's an improvement from last time in terms of degrees.

And then there's this fun figure for the crossover. So in general, that's just a repetition of what our XOR gadget was. Now we have crossing XOR constraints. So here's a typical example. Let's say we have this XOR trying to go from there to there and it crosses these other XORs.

So what we're going to do is build something like that gadget for the vertical connection with this, and this, and this, and this. And we duplicate these edges. That doesn't help you-- I mean it's the same as that picture, I just happened to

duplicate them. The reason we do that is we're going to put XORs across like this, and this forces an alternation.

So if this is in, then this must be out, so this must be in, so this must be out, so this must be in, and so on all the way through. And so that forces an XOR between this edge and that edge. Cool?

So that is the crossing x over XOR gadget, which you plug into all of these pictures, and you end up with planar directed max degree three. All right? So far so good.

Next-- I have a goal. I'm still not going to tell you the goal. Next one is going to be planar bipartite max degree three. Undirected. This is going to be one of our simplest reductions ever. There it is.

So the cool thing about max degree three directed-- I mean there are few cases you can exclude. You can't have three incoming edges. You can't have three outgoing edges. It would be pretty hard to have a cycle that includes such vertices.

So you could have degree two vertices, but I don't think we even had any. Did I say that we had degree two in this picture? Probably not. I mean if we did we could double the edge or something. So that was probably actually planar directed three regular.

OK, but now undirected. But you can basically have two edges in, one out, or one edge in and two out. And we're going to convert that. So the interesting thing here, you have to visit this vertex, which means you have to visit this edge in any Hamiltonian cycle. Because once you come here you've got to leave on that edge.

Similarly, you have to visit this edge. You can tell that just from this picture. So we call these forced edges. And we can represent that by adding a vertex there. It says, hey, you've got to visit that edge. So you're going to visit it by that little path there.

And now things are bipartite. And the bipartiteness is essentially forcing the directionality as well. So let's say when you come into a vertex and you're blue, so

that means these guys have had to have been red, whenever you leave, you first visit a red vertex and then go.

So here you're coming in again as blue, and then before you leave you turn red. And so because you've satisfied directionality over here, the outgoing edge here becomes an incoming edge to the next vertex, that means you'll be going into a blue vertex either because it's like this or like that.

And you are effectively forcing directionality. I mean everything could be reversed. Because you're always coming into a blue vertex and leaving on a red. That forces you to be going from the tail of an arrow to the head of an arrow. So same thing by symmetry.

So these problems are actually really close to each other. These were done in different papers, but this was an older one. And then this problem was introduced for another reason, which is grid graphs.

This is where I really want to go. So the idea with the grid graph is you, let's say, have a set of points on the square lattice. And then you're going to have an edge whenever you have two points at unit distance.

So let me draw an example. Cheat, and have some examples drawn for me. This was in a much more recent paper, but some nice pictures.

So we're just choosing a set of points in the square grid. And then whenever we have two points that are a unit distance apart, you're forced to have an edge there. You can't say there's no edge. That would be a sub-graph of a grid graph. But for example, when this point is absent, you don't get any connections through here.

So these are all examples of grid graphs. Now there's one special case, or in general, we distinguish-- there's sort of three types of faces in such a planar graph. There's what you might call pixels-- little one by one squares. There's the outside face, and then there's everything else which we call holes.

So any non-length for face we call hole. If there are no holes we call it a solid grid

graph. This is interesting because solid grid graphs you can solve Hamiltonicity in polynomial time.

It's not always possible, but there is a polynomial time algorithm for solving the solid grid graph case by Umens and Lenhart, '97. A very complicated algorithm. Luckily, this is a hardness proof class so I don't have to cover it.

I'm going to focus on the hardness when you allow holes. So this is where things get fun. We are going to reduce-- we did a reduction from here to here. We're going to do a reduction from here to here.

So we're going to suppose that we want to solve Hamiltonicity in planar bipartite max degree three graphs, and we're going to convert such a planar bipartite max degree three. Graph into a grid graph that acts the same.

So the first step is to draw the graph on a grid. That's pretty obvious. So here's an example of what we want to do, and then this is a picture of how to do it.

Suppose you have a bipartite planar graph. Not obvious this is planar, but it's obvious that it's bipartite. It happens to be planar. It's max degree three. And the vertices are labeled, the XIs and the YIs, and you can see the corresponding labeling here.

So not all of the grid dots are going to be labeled. Some of them are just auxiliary. In order to connect to vertices we're going to have to draw paths, orthogonal paths on the grid.

But the key thing I want is the square grid has a two-coloring inherently, and I want that two-coloring to match this two-coloring. That's what I would call a parity-preserving embedding of the Graph

And you can see in this example, the three XIs are white, and the three YIs are black. And if you check all the edges they should correspond. Just ignore the dash lines-- that's to show you the grid underlying.

So this is a valid embedding. How do we do it? Well, we just take any embedding in

the grid. It's well known that if you have a max degree four graph you can draw it in the grid without crossings, and each edge is some orthogonal polyline. I won't prove that here.

And then suppose the parity's wrong for some vertex. Suppose you draw vertex B and it happens to be black, but you want it to be white? Then I'm just going to shift it over by one and do these kinds of wiggles, and that's it. So you scale everything by a factor of three and you'll have room to do those wiggles.

So take an arbitrary orthogonal grid drawing and then add the wiggles whenever things are in the wrong parity and now every vertex will be in the correct parity. That will be crucial for the reduction.

So at this point we've got an orthogonal parity-preserving drawing of our given bipartite planar max degree three graph. And the issue is there are sort of two types of vertices here. There's the ones there we really need to visit, and then there's these other vertices which are representing edges.

We don't need to visit every edge-- that's the tour problem. Hamiltonian tour, we just need to visit every vertex which are the labeled dots over here.

So this is where things get fun. Here is a key gadget before we see the overall architecture. These are called tentacles in the original paper. I'm calling them edge gadgets because they're just simulating a single edge of the original graph, but tentacles is cooler. Whichever you prefer.

So these are the dots that haven't all been drawn explicitly here. And so it's just like a two by and sort of corridor. The base version would be this, but if you want to have turns, which we're going to need 90 degree turns, it looks more like this.

And there's basically two ways to cover this. Sort of three ways. But the way parity will work out, there will only be two ways that we care about. There's not a variable gadget. It looks like maybe a variable gadget, but that's not how we're using it.

So one way you can visit this gadget is to zigzag like this. We've seen this in pretty

much all of our planar Hamiltonicity proofs. The core idea's the same. You could start at A and end at C, or you could start at B and end at D, and that's true in this version as well.

Once you start alternating you're forced to alternate all the way through. So you could imagine using that for a wire, but we're not going to use the two alternations. By parity we'll be forced to do it one way or the other. We'll be told whether we're starting or ending in one of them, I think. Well, it won't matter.

But there's this other way you could visit, which is to just go all the way down then come all the way back. This is going to correspond to an edge that is in the Hamiltonian cycle. Because I want to start at some vertex up here, and I want to end at some vertex down here if I follow the edge.

This is going to correspond to an edge that we don't use. Because at this vertex, I still need to cover these vertices somehow, but I don't want that to really matter. I want this edge to be coverable for free. So the idea is as I visit this vertex we're just going to zip down here, zip back up. And all these vertices get visited for free. It's not really part of the tour.

Let me show you in some more detail. Here's what a vertex looks like. Vertex is just going to be a 3 by 3 grid of dots. Now, remember there are two kinds of vertices. There's the ones that were-- so again, we're expanding everything for this to work out.

The vertices that were originally white and the vertices that were originally black. It doesn't matter which is which, but there's two classes of vertices. And we're going to attach the edge gadgets, which were these 2 by n kind of things. So this is all edge gadget here.

We're going to attach it to a vertex that's white in this sort of flush length two connection. So this is the variable up here, this 3 by 3 thing, and here we're having two edges connecting to the edge gadget.

For the black vertices we're only going to have a single edge connecting them. This

is called a pin connection, between the wire gadget or the tentacle, and the 3 by 3 down here. OK. Why? Because.

So here let's consider two vertices that are attached by an edge. So they have opposite parity. One's-- I've forgotten which is which. This is white and this black I think. Here we have the pin connection. Here we have the length two connection.

And the idea is there's exactly two ways to traverse this edge, as we saw. But now you can see the connections. Let me put it that way. So the idea is, well, maybe you choose to use that edge in the cycle, and so I want to be able to just traverse it. Everything is good.

And so here I know that I'm entering at this point, which means I know that I'll have to be exiting here. So the parities are forced. So the zigzag parity is forced.

The other possibility, because there's only one connection here, you couldn't traverse this way and then come back. But from this side you could traverse and then come back. And it doesn't touch the vertex at all.

So the idea is you could ignore this edge entirely and it gets covered for free, provided once you erase this, this edge is in the Hamiltonian tour. And at this point I should mention what all this labeling means on the vertex gadget.

The idea is for this vertex, if you start at any of the PIs, any of the four corners-- any of the private investigators-- then you can end at any other PI, PJ in such a way that you visit all three of the EIs.

AUDIENCE: There are four of them.

ERIK DEMAINE: There are four of them. EI, EO, EIEI-O. No, OK. So let me get to the point. You could start at PI, end at PJ, and visit all the vertices and all of the EIs, all of the EKs. So for example, if I want to go from here to here, I can go like this, visited all the edges. If I want to go from here to here, I can go like this, visited all the vertices, visited all the EIs.

So that's good because each of these EIs is potentially attached to an edge gadget and we might want to visit all of the things out there and come back. OK?

So you see the reason why we only want a pin connection here is if we had a full connection with two edges, you might imagine that you come out here, do something, eventually do other stuff, and then come back on this path. But because we have a pin connection always on one of the two sides, you can't have two separate traversal paths.

As soon as you get here you have to turn back around. If you went this way, then this guy would never be visible in a cycle. So that's the key thing. This was one of the EIs, and so the idea is you can completely forget about this path. You can just think about going from one of these white vertices to another. In general, it's going to correspond to which edge gadgets that vertex gadget is connected to.

So you can go from one vertex to another. You will visit this edge. And so you can add in this part if you don't use the edge. If you do use the edge, then you'll be starting from one of these white vertices and you want to go to this one. And then you'll follow this path. And then you get to this vertex, then you'll go from one of black vertices to one of the other four corners, and so on.

Question?

AUDIENCE: Should that line of three white dots, shouldn't they all be black?

ERIK DEMAINE: Yes, this should be black. It's not my fault. It could be the scanner or it could be-- this is a hand-drawn figure as you might guess. So yeah, that should be black. Everything should alternate.

So you can tell the parity's a vertex based on the color of the corners. Any other questions about that proof? OK.

So this is pretty cool. And I would say Hamiltonicity in grid graphs is used a lot as a base problem for reductions in computational geometry, especially tour-like problems. And you'll notice a theme.

Every reduction so far, except the very first one I guess, was from one type of Hamiltonian circuit to another type of Hamiltonian circuit. And this tends to be common that you can work within Hamiltonian circuit and modify your graph in all sorts of ways to get to a very simplified version of Hamiltonian circuit. And that's cool. And so you get a lot of mileage just within the Hamiltonian circuit family.

Cool. Let me give you a couple of applications right off the bat. What?

AUDIENCE: This is going back a little bit, but are there any sort of natural families of graphs for which finding a Hamiltonian path is strictly easier than finding a circle?

ERIK DEMAINE: I think the only-- this problem would be easy. Find a Hamiltonian path given a Hamiltonian path. But I'm not aware of any other problems where a Hamiltonian path is easier than cycle. I mean in general, you can reduce one to the other somehow. That's maybe a little tricky without the two given end points.

And if you have lots of structure, like planarity and through regularity it's not so clear, but definitely Hamiltonian path is still hard in this problem. I think one easy way to do that is to just do the same chain of reductions, but start with Hamiltonian path instead of cycle and follow through, and pretty much every step should still be path-based.

So it's actually useful. We'll want Hamiltonian path and grid graphs also at some point. But what I showed was Hamiltonian cycle. So for pretty much everywhere they seem to be the same, but it would be cool to find an example where they have different complexity.

OK. So here's another problem. Euclidean TSP. Traveling salesman problem. You're given a set of points in the plane. Better not make it too big because this might be hard. And I want to find a tour. It visits every point exactly once, and has minimum total length, Euclidean length. So the usual notion of length.

I claim this is NP-hard. Now, this is a little scary. Most of the problems we've been working with are very discrete. I mean the output here is sort of discrete. You

choose a permutation on the points. But it feels like there's almost no control here.

I mean how would I construct localized gadgets when I can just jump from any point to any other point at any time as long as that point hasn't already been used. But it's really easy to prove this is NP-hard given what we just did.

Oh, I should say, here's a full worked out example, so this is what I should be pointing at. This was the parity-preserving graph that we drew before. And this is what you get in the reduction, along with actual solution, which is the one drawn in orange here.

So you can see read that off, because wherever you see zigzags, that's where you're using the path here. So these correspond, and then this corresponds. And then up and over. And then the unused edges, the not orange ones, that's where we get these kinds of loops, starting from the vertices of one parity, I guess the white ones, and then going to the black ones and turning around at the pin joint. Cool.

So given this is hard, why is this hard? So the decision problem, let's say, is I give you a set of points. I give you the desired length of the tour. I want to know is there a tour, visits every point exactly once, and has length at most that given length.

AUDIENCE: So if you look at a grid graph, you'll have some number of nodes, which means the tour would have to necessarily have-- it can just choose those points. The tour would necessarily have at least-- exactly that many edges, and each edge would have to have length at least one, because the close [INAUDIBLE] points is one apart.

ERIK DEMAINE: Yup.

AUDIENCE: And so if you just counter [INAUDIBLE] those points and say that that's the desired length of your tour, then you can only do that if you can get through this by following only edges that are length one.

ERIK DEMAINE: Exactly. So we're going to reduce from this problem. Given such a grid graph, the

reduction just looks at the vertices, throws away the edges. That is our input-- those points are the input to TSP.

And then the claim is that every tour-- let's say there are end points in this picture. Every tour has to have length at least n, as you say, because they're end points, the total number of edges is n in a cycle, a Hamiltonian cycle, and every edge must have length at least one, because that's the nearest pair. And so that's a lower bound.

And then on the other hand, if there is a tour that has length only n, then it must actually be a Hamiltonian cycle in this graph, because then you're only using edges that exist in the grid graph.

So it's really the same problem. Euclidean TSP really is a special case of this. Cool.

This problem was proved hard much earlier. Hamiltonicity and grid graph. But here is now a nice simple proof using what we know.

I'll tell you more reasons in a little bit, but I have one more at this point, which is platform games. I don't have a graphic because there's so many games that could apply here, Super Mario Brothers among them.

Suppose you have a little dude, you can walk around. And you've got some kind of coins-- I don't know how to draw coins. They almost look like eyeballs. But anyway, you've got a bunch of coins which you want to collect. And you have some way that you can walk around. If you're a top-down game, then that's pretty much the entire reduction.

And then also you have a timer. So timer plus collectibles-- ables or ibles? Ables. And any such game is NP-hard, at least, because if you want to get all of these coins, that's like vertices you need to visit. As I say, a top-down game you're walking in a grid graph.

The time limit will prevent us from going out into the white space because there's no coins there. And so if the timeline limit is exactly the number of coins, and every

step you make you have to get a coin, then that will force you to only follow-- the edges in the grid graph can't go out here. No obstacles needed.

And you understand that in the real game, of course, there might be stuff happening here. Maybe there's a ladder so you can go up here. You just have to set up all these traversal lengths to take the same amount of time in under optimal play. And then you can simulate a ton of video games in this way.

This is a result. This is in Fun 2010 by Forishek. So some reasons why you care. A lot of games map naturally onto grids. A lot of real problems can map onto grids. In general, grids will be very helpful.

So next problem I want to solve is-- by solve, I mean show hard, as usual. Let's say, max degree three grid graphs. I don't have a great motivation-- well, I have a motivation for this. Probably others that I'm not aware of. I didn't actually know about this result until recently. It's pretty cool.

If you look at this reduction, there are some vertices that-- I mean here we're drawing the cycle, but some vertices have degree four. Of course, every grid graph is max degree four. And max degree two is pretty easy to solve Hamiltonicity. So what's left is max degree three. So a natural question in between is can you do max degree three grid graphs. The claim is that is also hard.

And it's going to be basically the same proof, but we're going to change all the gadgets, most of them. There's one shared author. This was Eti Papadimitriou and Szwarcfiter. And then this is Papadimitriou and Vazirani a couple years later.

So this is actually very similar to the edge gadget we saw before. Here's the old edge gadget. Before when we had a turn we had a degree four vertex. I want to get rid of that, and I'm going to get rid of it by just sort of shifting these things apart and blowing up this circle, this cycle.

So I end up with a bigger cycle here, end up with an edge connection there. But effectively, it's the same kind of topology. If you're coming in and alternating-- I guess you have to get the parity right.

If you're coming in from this side and you're alternating, then you can come around and visit like that, and everything is OK. Back and forth. Cool. That was drawn right over here. That's a zigzag path. The return path is just as before. And so we can use this edge gadget exactly as we did before, but now it's max degree three everywhere.

So that's easy. In general, using this trick that here we had a corner with a vertex on it. You can turn that into a longer path with various vertices on it. That will act the same from a Hamiltonicity perspective.

So that's the key thing we're doing. And then it's a matter of putting things to fit on the grid. So that's the first change.

Now, the second change is going to be a little bit more drastic, a little weirder. The vertex gadget used to be this 3 by 3 thing. Very hard to get rid of that, vertex would have to be four in the center.

Well, if you did that, you would be left with a little square, an empty square. And we're going to need two of those. So this is the dumbbell picture used in many theoretical results. So we have two of these squares. We're going to separate them by a fairly long path of the right parity.

And the idea is you can have up to two connections over here to edges, and up to two connections over here for edges. And typical connections are going to look just like we had before, but it's not going quite all be like this.

So it used to be we could have a full edge connection or we could have a pin connection as before. When this edge turns it's going to use the weirder turn gadget. So the edges look basically the same, it's just the turns that we changed in the last slide.

OK. This almost works, but I have to specify some details. So for example, easy case. Degree two vertex. Suppose the original graph has a degree two vertex-- we had that in our reduction.

Because in fact, every other vertex pretty much was degree two, if you track all of these things. When we went from the directed case to the bipartite case we added in lots of vertices, lots of degree two vertices.

So if we want to simulate a degree two vertex, we will just have one of the edges attaching up here, and one of the edges attaching down here. And again-- in this picture it's one on the left side, one on the right side. And again, we have this feature that if we start at any of the PIs we can get to one of the PJs by visiting all the EI edges.

So for example, if we start here we go like this over here. And we have a choice. We can either go like this, end at P3 or go like this and end at P4. We visited all the EIs.

But we can't start at P1 and end at P2. We want to visit everything. Start at P1. If you go out there, you can never come back. To come around here, you've already visited P2.

So we cannot start and end on the left side. We cannot start and end on the right side. So for a degree two vertex, that's fine. We just put one of the connections on one side, one on the other, and we'll come in, visit everything, come out. And because we visit all the EIs, we can do this kind of traversal every time.

Now, for a degree three vertex, this is a little bit troublesome. Remember our picture. What do degree three vertices look like? If you follow through the chain of reductions, we had this case, and we had this case, and this one we converted into that. And this one we converted into that I think. Yup. Because this was a forced edge. This was a forced edge and so we added a dot to represent that.

So the good news is every time we have a degree three vertex in the starting graph, one of the edges is forced. One of the three edges is forced. So it's really a choice between do you take this one or do you take this one. I mean that's, of course, what we had over here.

So what we're going to do is take these two edges, which are exclusively OR'd

together. I choose exactly one of these. We're going to put that on one side, so that one of them will go up here, one of them will go down here.

And then the forced edge we'll put over on the other side. Forced edge is always chosen, so that means we'll come in either P1 or P2, or leave on P1 or P2, depending on which of those two cases we're in. And then we will leave or enter from P3 or P4, according to which of the two edges you choose.

Happy? No?

AUDIENCE: I'm understanding that we're actually reducing from a sub-problem of [INAUDIBLE] bipartite, max degree three graphs in that every vertex is connected to at least one degree two vertex?

ERIK DEMAINE: Yeah. Right. So we could say at this point-- in fact, it's planar bipartite max degree three graphs where every degree three vertex is adjacent to exactly one, or no, at least one degree two vertex.

And then we can carry that condition through all the reductions, and that would make this an actual reduction. Or you can think of this as a reduction going from here, just copying all the details. Yeah.

Other questions? OK.

I'm a little confused because there's one more gadget. But I didn't seem to use that gadget. So I'm not sure whether this is necessary. But here's another thing you can do, which they're using for the forced edges.

You can have one wire-- sorry, this is the picture. You can have one edge gadget that attaches to both ends of the dumbbell. So then you have a choice whether you end up connecting that edge to the bottom of the dumbbell or to the top of the dumbbell. Sorry. This is connecting to the top, and this is connecting to the bottom, the dumbbell.

So that's another way to do the same kind of degree three vertex. You could have this guy coming in-- this is the forced edge, and the forced edge has to come to this

vertex. You get a choice of whether you start up here and then leave along this non-forced edge, or you first traverse down this way and then leave along this non-forced edge.

So that's another way to do it, but it seems just as good as the way I described, which was to put forced edge only a one side, and then the other two guys on the other side.

Unless there's a parity issue. It could be that this connection and this connection have the opposite parities, which is not good. Because we want the two connections-- in this case, both of these are going to be white. And in this case both of these are going to be black.

So the two non-forced edges connect to vertices of the same color. And I think if you try to attach one of them over here and one of them over here on a variable, they will have the wrong colors, they'll have opposite colors. I think that's why you need this gadget.

Because now this left connection and this other left connection will have the same parity because this count is odd I think. I think that's why you need it. And that preserves parity. So it's a bit subtle. You have to be careful all the connections preserve parity. Cool.

So that was max degree three grid graphs. Any other questions?

Here is one motivation from this Papadimitriou/Vazirani paper for this problem related to Euclidean traveling salesman problem. It's this notion of degree-bounded minimum spanning tree.

Minimum spanning tree normally is a polynomial problem. We're given some graph, you want to find a spanning tree. Tree that this is all the vertices, but it can be a tree instead of just a path. And it's minimum in that it has minimum total length of all the edges. So that's a polynomial problem.

If I say, well, I want a minimum spanning tree where all of the vertices in the tree

have max degree two, that is Hamiltonian path, or I guess really traveling salesman problem. So we already proved Euclidean max degree two minimum spanning tree is hard. What about max degree three? Does that somehow-- does the branching make it easier, and the answer is no.

You start from this problem. And the cool thing about a max degree three grid graph is if you look at any point, at most three of its neighboring spots are actual vertices. One of them has to be absent in a max degree three.

So what I'm going to do is add a point really close to that one. And now you can show in any minimum spanning tree, these guys must be connected, and this one basically has to be a leaf because everything else is quite far away. I mean at most you have these guys occupied. You know that this is completely absent.

So I add this point. I'm going to have to connect this. it won't be able to connect any other way if I'm using the shortest possible length. And so at that point, if this has degree three, there's only two incident edges left, and so it's Hamiltonian path again in the grid graph.

So this is why they were motivated to define max degree grid graphs and prove that hard. Cool. Some more geometry.

All right. So for a long time-- you know, this is all done in the '80s, all the grid graph stuff. We only had square grids. We couldn't afford triangles back in the day. But recently, people started to think about triangular grids and hex grids. Those are the only tiling regular n-gons.

So you can define the same kind of thing. A triangular grid graph, a hexagonal grid graph. Again, you take a set of points in the triangular lattice or the hex lattice. Is the hex actually a lattice? Well, anyway, you know what I mean, the hex grid. And you connect any two points when they're a unit distance apart, where that's defined to be one and one.

So again, we have the notion of solid, triangular, or hex grid graph, and there are no holes. Again, we can characterize the pixels-- they're not really pixels-- hexels and

trigles? I don't know. In the paper they're all just called pixels. There's those types of faces. There's the outside face. And then there's the hole faces-- H-O-L-E, not W-H-O-L-E. OK. Cool.

They define some other terms, which I will get to. Let me tell you what's known about all of these problems.

So first thing, so these are-- So first thing that's known is that if you have a solid grid graph, then the triangular case is also polynomial. So I mentioned solid square grid graphs are easy to solve Hamiltonicity in.

Turns out solid triangular grid graphs are also easy to solve grid graphs-- solve Hamiltonicity in. Almost all of them are Hamiltonian in a certain sense, as long as you check for some basic things you shouldn't have.

But hex is open. So there's actually a ton of nice open problems still. Next case is called a super thin.

So what is super thin? Super thin means there are no pixels. Every face is either a hole or the outside face. I don't have an example here, but this kind of connection is what you would have in super thin graphs.

So you've got the whole face here, a whole face there. You shouldn't have any of these pixels to be super thin. That case turns out to be polynomial for squares and hexes, open for triangles.

All right, next case is called thin. That one right there implies. In the thin case, both the triangle and the square, problems are NP-hard, hexes are open. What does thin mean? Thin means every vertex is on the boundary.

So super thin is something like every edge is on the boundary. Thin is that every vertex is on the boundary. So for example, this part here is thin. The banner's actually drawn in bold here. If every vertex has bold edges incident to it, then we call that thin. So it's a little bit thicker than super thin.

So it's also easier to think of the dual if you care to. If you draw a vertex for every pixel, in the super thin case there are no vertices in the dual. In the thin case, the dual is super thin. I don't know if that's easier to think about.

But these cases are actually hard, at least in two out of the three pictures. One of them we have already proved. If you have a max degree three grid graph, if max degree is three then it must be on the boundary because of that absent pixel.

So we already showed the thin case for squares is hard. Max degree thin implies thin, four squares. And so you might ask what about triangles?

OK, so finally we get to an actual proof. It's going to be the same kind of thing. I'll skip this detail, but this is the planar and betting part. So again, we're going to reduce from bipartite max degree three planar Hamiltonicity as we did before.

And then we're going to draw it in the plane in some kind of parity-preserving sense with edges routed on the triangular grid. That's not hard.

Then we have very similar looking gadgets. Although in this case, a vertex can just be a little triangle of dots. The edge gadget's going to look pretty similar. It's like a ladder, but now the ladder has funny rungs on it. Turns are just what you would expect. Notice we get degree five here.

But otherwise, things are the same. You can traverse either in the loopback configuration. When you have this pin joint there you'll be forced to turn around there. Or you zigzag all the way through, and then that's used for present edges. So this is edges you don't use in Hamiltonian cycle. These are edges you do use.

And there are two types of vertices. There's the ones of one parity where you always meet-- here, you're kind of meeting with three edges, but you're kind of meeting on a whole edge, like that.

And from here you have the choice whether you do this or do this. Two of them will be zigzags, one of them will be double-back.

And then there's the other parity class where you only have pin joints. These are,

again, just connected by single edges, so they act just like we had before.

So pretty much the same reduction, but with different-- in some ways, this is an easier reduction I think. Cool.

Now, that was max degree five. What about max degree four? So here we had degree five vertex there. If we kind of do our same trick of spreading them around, you end up with degree four at most. And it acts the same. And the vertices we're going to explode into a hexagon. And it works.

So I should say max degree four triangle is NP-hard. This reduction should also be thin. Previous one may have also been thin, but again, all of the vertices are on the boundary. So this is the thin triangle is hard. Max degree four triangle is hard. We did this one. So we've done all the hardest proofs.

That will cease to be the case in a moment. OK, here's one more thing. Max-- so what about hex grids. I haven't mentioned any hardness for hex grids yet. And one thing we know is max degree three, writing new squares, but also hexagons are NP-hard.

Now, this is not so impressive. Every vertex in the hex grid has a degree at most three. But you can't hope for two, so that's the best you can hope for in max degree characterization with respect to hex grid.

This proof is tricky. Maybe before I go there let me just mention one other result. So this is delayed.

One other notion you can think about for grid graphs is called polygonal grid graphs, and that's what's supposed to be illustrated here. Polygonal's kind of the opposite of super thin. So super thin we had an edge that had non-pixels on both sides.

In polygonal, every edge has a pixel on at least one of the two sides. So you never have an edge between two holes or a hole in the outside face. That's a polygonal situation.

So that's something you might assume. It's sort of like saying you had a polygon and then you sort of took the interior of the polygon, whereas here, this was not really a polygon because it's like there was a doubled edge there or something. That's not really a nice picture.

So polygonal's something you might hope for in the practical applications, which we will get to soon. Funny thing is for polygonal triangular grid graphs, the problem is polynomial. And for polygonal hex grid graphs, the problem is NP-hard. And for polygonal square grid graphs, we don't know. Another open problem.

I won't prove this. I think, again, this is almost always feasible. It's almost always a Hamiltonian path. For this, I'm guessing it follows from the same proof.

Let's go into this proof, hex grid hardness. Again, there's a drawing step. You're given some bipartite max degree three graph, and bipartite planar max three degree graph. And then we're going to draw on the hex grid.

In this case, we're going to represent variables by sort of horizontal strips, just because that's what is a little bit easier to do. So it turns out-- I think turns are really annoying on the hex grid. Hard to do a turn. So we're going to focus our turn attention within these little rectangles.

And it turns out you can draw everything in such a way that all of the edges are straight lines, straight line segments, if you grow the vertices to be these longer horizontal things. So believe that.

Then main idea is a very similar kind of thing. In this case, they're going to draw the pictures and they're going to draw all of the forced edges that must be in there, and here they haven't drawn the part in the middle. There's two ways to fill it in.

Essentially, you could zigzag back and forth, just like before. Or you could go there and come back, just like before. So edge gadget looks pretty reasonable.

These are some more gadgets. These are like sub-routines they're going to use. Their named and they have color patterns to give you a sense. This is going to be

the center of a vertex, and there's a few different things you could do with that.

This is some kind of turn gadget. That's what I called it. They called it something a little different. I think a rosette. But you can sort of zigzag, or you can kind of go there and double-back, but you end up with different orientations at the end.

This is what they call a U turn. Again, these edges are forced. You can either use it as a zigzag, or they could both kind of cancel each other out. Sort of don't meet each other. So you could do different kind of connectivity.

With that in mind, here is all the vertex gadgets. So I don't know all the details here. But I think there's a lot of cases because of the different ways things could be connected. You could have like one on top of the-- notice the dash things are the rectangles that I had before.

Normally they'd be really spread out. So maybe you have two in the bottom, one on the top, here I have two on the top, one on the bottom. And they could be oriented like this, or oriented like that. So there's a lot of different cases as a result.

The core part in the middle is the same. It's pretty much like the examples we had before of essentially like a little 3 by 3 square for the square case, or a little triangle in the triangle case. A little bit bigger in a hex. It's basically a cycle is an extra dot in the middle.

But then this was a U turn gadgets, so if something was coming up here, you could just sort of turn back around if you don't want to use the edge. I think these are prob-- well, yeah, so there's probably also two parity classes. There's the turn gadgets. And it works, but it's complicated.

I think that's all I'll say about various grid graphs.

AUDIENCE: Yeah. You said that in hexagonal grids were open? Did you mean the vertex? Every vertex on the boundary--

ERIK DEMAINE: It means every vertex on the boundary.

AUDIENCE: --[INAUDIBLE].

ERIK DEMAINE: No. It's not necessarily the same. So if you have hex grid-- I mean this could still be a boundary cycle and you still have a degree three vertex. Something like this part is thin. And I'm guessing you could complete that.

So they're probably very special, so I could imagine the thin hex grids are polynomial, but it's not known. OK.

One more picture. This is an actual explicitly constructed example from that graph to this crazy gadgets. But now you can see a little better what the rectangles look like. If you want to stretch something out, there's always horizontal wires that you can stretch out.

And I keep calling them wires. So those horizontal pieces of edge gadgets that you can stretch out so that you end up firing and hitting the next vertex, which is this rectangle. And when you spread out the rectangles you have to stretch out these diagonal shots. Cool.

Those hex grid graphs. Why do we care about hex grid graphs so much? Because Settlers of Catan, and various other board games that have a hex grid.

Now, has anyone here not played Settlers of Catan? OK, a few. There's a whole game to it. But--

[LAUGHTER]

What I'm going to focus on is this card. It's called longest road. So in general, you're constructing cities. Don't worry about cities. But you're constructing these road segments. They're edges on the hex grid.

And the longest road says that you get two points, two victory points, if you have the longest road, meaning the longest path that can be formed using edges of your color versus edges of anyone else's color, then you get two points.

So here is the idea. Settlers of Catan has randomness and stuff like that, but again,

if we look at a mate-in-1 problem, that goes out the window. Just can I win in this one move. There's no randomness to that.

Let's suppose you have tons of resources. This is what you need to build a road. And inconveniently, your opponents have built cities and things in all of these white spots, so you're not allowed to build roads there. That's just part of the rules. So that's the opponent's form obstacles.

And then because you have infinite cash, you can basically build the entire grid graph here. And then to figure out whether there's a really long road, namely a road length n, where n is the number of pixels that were not occupied by opponents. Number of dots that are not occupied by opponents is NP-complete. Cool?

And your opponent has a slightly shorter road, so you really have to distinguish whether you had the longest road. In fact, I like to say mate-in-0 is NP-complete, deciding whether you've already won the game is NP-complete.

Now you don't even need cash. It's just I give you a configuration. Do you have the longest road is NP-hard. So it's the only mate-in-0 hardness result I know of. Maybe we can find more.

OK, so that's why hex grids are cool. Let's do some more games. Here is another Nikoli puzzle, Slither Link. Their more famous one.

So you're given a grid of squares. Each square is either blank, meaning no constraint, or there's a number saying there should be exactly two edges among those four. And there should be exactly three edges among those four.

Your goal is to find a cycle-- not Hamiltonian but just a cycle, doesn't visit all the dots-- that satisfies that neighboring constraint. So here there were exactly two. Here there are exactly three edges in your cycle. OK? This is the very first puzzle on the website. Again, there are books on Slither Link puzzles.

And here is a little overkill of a hardness proof, but in one slide. So this is a reduction from Hamiltonicity in planar graphs. So you take any planar graph and

draw it in the grid. This is one of the very first steps we did today.

And then if you do that, there are two types of vertices. There's the ones you have to visit and there's the ones you don't have to visit. Now, we know grid graphs are hard. So we could actually assume all of the vertices have to be visited.

But for whatever reason, this proof did not do that. This is back in 2000, so at that point maybe it was not so well known. But these days, grid graphs, Hamiltonicity is used a lot, but probably not so much then.

So anyway, there's this gadget which is you can do whatever you want, basically. These are the vertices that don't need to be visited. The key gadget is this one. So let's focus our attention here.

We have these ones which means the cycle must come here. And so collectively these are going to form Hamiltonian cycle constraints. You're building a cycle and you have to visit these ones. Now it turns out no matter which end you come in and which end you leave, you can visit all of the ones. You could be adjacent to all of the ones. So different turns are going straight.

And so if there is a Hamiltonian cycle, you'll be able to visit all the ones. And conversely, if you can visit all the ones, that must be a Hamiltonian cycle.

The one extra thing we need to say, if you just glue these gadgets together, then there's no-- you can go from anywhere to anywhere. So you just add in some zeroes here if you don't want there to be a connection from left to right. So you can use that to delineate the boundary.

So this could actually do a sub-graph of a grid graph even more special, or you leave a clear if there's an edge. And so you get a picture like this. In this case, they're using these bunch of zeroes to say, well, you don't have to visit these pixels because they're just simulating a big edge. But we really have infrastructure to simulate edges. We don't really need that.

So that's a really easy proof. I would say in general, what you have to be careful

about here is that edges are free, because in Hamiltonian cycle you're not supposed to pay for the edges. You just have to visit every vertex. But that's really easy in this kind of set up.

Another Nikoli puzzle, Hashiwokakero, bridge building. You have a bunch of dots in the plane. Each dot has a number on it, specifying a degree request. And your goal is to make a connected structure. It looks like it's usually a tree, but that's not part of the constraints.

Where every vertex has that degree, you can use multiple edges. You could build multiple bridges between the same two cities. And so there's four incident to that, eight incident to that, and everything's orthogonal. It can only build horizontal, vertical. You can't build a bridge like this. You're not allowed to make turns.

AUDIENCE: Same as standard coding where you've only got two parallel edges per [INAUDIBLE].

ERIK DEMAINE: Ah. OK. So also standard is that max double-edge, no triple-edge. Thank you. Obviously, i haven't actually played. It won't matter in this reduction. So here we are reducing from Hamiltonicity in grid graphs. So these are edges that we might want to allow.

In general, we're going to use these little ones to delineate the boundary. It's just the same kind of trick we use here with Euclidean degree three MST. We added a little dot just off to the side.

It's not a metric problem here, so we're just adding it one unit over. That delineates the boundary. It basically is forced to connect into the only neighbor. If you tried to do this kind of connection, you'd end up being disconnected. You'd just get a single edge there. Because ones will only face ones.

Sorry. The other constraint is you're not allowed to have crossing bridges. That won't matter here.

So we delineate the boundary at the ones, and then every other vertex is going to

have value two plus the number of boundary edges incident to it. So here it happens to be four, here it happens to be three.

So I could have gone like this. That would have been fine. But in the cycle, I happened to go around that way. That's representing the cycle.

So again, very easy. This shows you how useful it is once you have grid graphs. You can do any problem on any grid, pretty much, that has any kind of tour-like thing. That will be hard using these reductions. So that's why we care.

All right. Here's some practical motivation. Well, I have a game here. But suppose you're mowing your lawn. And so usually a lawnmower has a circular cutting tool, but in this game it happens to be a square cutting tool. It's basically the same.

And you have some polygon with holes that you want to mow. There's actually two versions of the problem. And they were introduced by Arkin, Fekete and Mitchell, 2000. One's called Lawn Mowing.

So Lawn Mowing you're given a polygon. You have to visit, let's say, if we're using a square cutter, you have to visit every pixel of the polygon. You have to visit the entire-- you're moving a unit square around. You've got to visit everything to cut the entire lawn.

But you're also allowed to leave the polygon, because for a lawn mower you can go outside the grass. You can go onto the pavement. Most lawnmowers that doesn't break it. So that's called Lawn Mowing. You can go outside if you want.

And the other problem is called Milling. And with Milling, you can't go outside. That video game is probably some hybrid between these two problems, but both problems have been considered a lot, especially from an approximation standpoint.

But there are lots of situations where this kind of problem comes up. Suppose you're laser cutting. This is one of the first laser cutter projects we did with George Hart, back in 2003, where cutting out these two-headed salamanders like that and then assembling them. If you've ever been to G6 in Stata, it's hanging there.

So you can think of this laser cutting problems, you have a lot of dots and you have to cut them all. That is literally what happens to laser cutter is pulsing with little dots. And in this case it looks pretty clear how to connect them together.

But if you're cutting many salamanders out of one sheet, and you think of the salamander's pretty small, that's like a TSP problem or really a milling problem. You have to visit all these things using the shortest-- it's really a lawn mowing problem. So you can turn off the laser and you can move around somewhere else that you don't want to visit.

3D printing is like lawn mowing problem in each layer. So 3D printing is done by printing each layer and then stacking them up one at a time. So in each layer you have to deposit material in FDM, say we're additive printing. You have to deposit material. If you're doing laser curing, same kind of thing. How do you move the laser with the shortest amount of work.

It's called milling. I don't have an image for this. Because if you're doing NC milling you have a drill and you're moving the drill around to cut out some shape or to cut a region.

There you're not allowed to go outside the shape you want to cut, because you go outside, you cut too much. And you don't want to have to pay to retract. There are other versions of the problem where you can retract the cutting bit and so on.

So of course we proved milling is hard. That is exactly the Hamilton [INAUDIBLE] graphs if you're not allowed to leave the region. Of course, that also shows that lawn mowing is hard because you never want to go outside the region.

One fun result is lawn mowing is hard even when you have no holes in your region. So if you cut these little slits, from a lawn mowing perspective it basically makes no difference because you could still go over it, and it doesn't really save you anything to not have to cut that. So lawn mowing is hard even without holes.

Cool. I have one more problem. Do you want to see it? OK.

So here's-- sorry-- a brief warm-up. Suppose you have a grid graph. You rotate it 45 degrees. You can represent each of these vertices as a vertical segment if they're black parity or horizontal segment. If they're white parity such that segments intersect, if and only if there's an edge joining.

So these are unit orthogonal segments. This is called an intersection graph where the vertices are the objects, and the edges are whether things intersect. I need that, so Hamiltonicity in these graphs is hard.

Here's a problem called minimum turn milling. So a big issue for 3D printing, for milling, usually when you're cutting or you're depositing material, you can move really fast in a straight line. But when you need to make a turn you have to basically come to a stop and then speed up again.

So suppose you have infinite acceleration, so to speak, or close to infinite acceleration. So once you're going straight you can go arbitrarily far for the same cost, and what you really pay for is the slowing down and speeding back up at every corner.

So then you want to minimize the number of turns, not the lengths. This is one of my first hardness proofs that was not about puzzles. I think the original paper was 2000, the journal version's 2005.

So suppose you have one of these rectangle intersection graphs. What we're going to do is replace each of these things with a cycle, and then thicken that cycle. This is a thin grid graph, not super thin. Maybe super thin. Here it's super thin, and here you have a little square that you're cutting with.

So each of these things you have to make four turns. Then also, to go from one of these to one of these, to go from one vertex to the next, you need a turn. So in general, you're going to need 5n turns in order to visit n of these original segments. And you can do that if and only if there's a Hamiltonian cycle in the original graph, which was this picture.

Basically, if you come in one position-- so here's one of these rectangles you want

to visit. You come in one place and you know that you want to leave on some other place. So you double this edge-- you're going to cut it twice. That's allowed in this model.

And then you loop around. And the number of turns here is exactly one to come in, four to go around, and then one to come out. In general, it's one per edge in the cycle, and four per vertex in the cycle.

So again, Hamiltonicity gives us what we want. Even where we minimize the number of turns, we can simulate that by counting vertices as turns, basically.

And that's it for Hamiltonicity.