AMARTYA SHANKHA BISWAS: So today, we're going to look at approximation algorithms for the traveling salesman problem. So I hope everyone knows what the traveling salesman problem is. You have a graph, you're trying to visit every vertex. So you start at your vertex, you visit every single one, and you end at the starting vertex. And you want to do that in the shortest possible time, or distance, or whatever the metric is.

So, unfortunately, this is NP-hard. Also, the approximation algorithms for any constant approximation is also known to be NP-hard. So you should have gone over approximation algorithms in lectures. So, basically, let's say the optimal solution has some value of e, and if you want to guarantee that your algorithm will end up with a value which is within a constant factor of v, so less than cv, that is an approximation algorithm.

But for the traveling salesman problem, the constant approximation algorithms are also NP-hard. So instead, we modify it slightly. So, on the traveling salesman problem, we impose something called a metric. So the important relation here is this one.

So, first of all, let's go through this. So you have a distance metric for-- so xy are vertices. Your distance is always greater than 0, which is reasonable. You're undirected, which is this relation.

And you have the triangle inequality, which means that if you have 3 vertices, and you have distance like this, this distance is always smaller than the sum of these two distances, which is like real world-ish things that make sense, right? If this distance was longer, it would just take this path instead. So the distance by taking other node is always greater than or equal to the direct distance.

So, turns out, the Metric TSP problem is also NP-hard. So nothing very great there, but you can do a constant approximation here. And you'll go through two approximations today. The first, the simpler one, is a 2 approximation, and then, we'll improve this to 3/2.

So let's start with the first one. So before we begin, let's define a couple of terms. Let's define c of S. So what is S? Let X be a path. Or, rather, let's say S is a set of edges.

So if you have your graph, here, your set of edges could be something like this, this, this, this. So that's set of edges. And c of S is defined as the sum of the weights of all the edges. And this, actually, should be a multiset, because you can count the same edge twice. So you can count this edge three times, or how ever many times you want. So that's the definition.

So, now, we want to find a cycle which goes through all the vertices, and we want to minimize the cost of that cycle. So let's say your optimal-- this should remind you of the Hamiltonian Cycle problem. So this is, essentially, finding a Hamiltonian cycle in the graph of minimum weight. So this is worse than the Hamiltonian cycle problem. This is find the minimum weight Hamiltonian cycle.

So let's say there exists this beautiful Hamiltonian cycle. You know of minimum weight. Let's call that H star G. So it's the best Hamiltonian cycle on this graph, and that's the shortest path that your traveling salesman can take. And so the cost of that is [INAUDIBLE] if H star S G.

So how do we go about trying to approximate this problem? So think about what algorithms you've seen that sort of connect all the vertices of a graph, and minimize costs of edges. Does it remind you of anything polynomial that you've seen in the class, that connects all the vertices? Expands all the vertices? Exactly.

So [INAUDIBLE], minimum spanning tree. This minimum spanning tree is polynomial time, and it connects all the vertex. So let's take some graph. Let's just go with this one. So you have some vertices, and you make a minimum spanning tree out of them.

Now, clearly, this is not a cycle yet. But let's try to construct a path out of this minimum spanning tree. So, first, let's root it. So let's say we are rooted minimum spanning tree. Let's say this is the root. So you have three things going out of there. I think that's it.

So let's give these labels. So, now, what we're going to do is, the way we're going to traverse all the vertices, let's just do a DFS. So, in DFS traversal, you'll first see 1, then you'll go down and see 2, then you'll go down and see 3, then you'll go back up, see 2 again. Go back down, see 4. Go back up, see 2. 1, 5, 1, 6.

So, basically, you're ignoring the rest of the graph. You find your minimum spanning tree, and you follow all the paths. Follow them back up, and just do a DFS traversal. And then, you go reach back to 1. And then, you have this-- well, it visits all the vertices. It visits some of them more than once, which is a problem, which we'll deal with shortly.

But it visits all the vertices, and it's a cycle. So, now, the problem is, the traveling salesman problem does not allow you to visit vertices more than once. Because, if you did not have this restriction, you could shorten your bat length by going to a separate vertex and coming back, or something like that. So let's make that more concrete.

So, at least in this case, given this triangle inequality, I claim that you can just delete the duplicate vertices. So let's look at the duplicate ones. You have 1, 2, 3, this repeats. So you delete that. Repeats, repeats, repeats, and that's cycling back.

So how do you delete things? So, in this case, you had a path, right? You were going 1 to 2, and 2 to 3, to 2, and so on. Let's write the 4 in. So this is [INAUDIBLE] only at tree edges. So how would you delete this?

So, let's say, you find the first duplicate vertex, and you don't want that. That's not allowed. So all you do is, simply, you follow the path, and you bypass it. And by the triangle inequality, you know that bypassing it will never increase your cost. It'll decrease it, or it will keep it the same.

So you can remove the duplicate vertex in this path just by bypassing it. But, also, remember that this is a complete graph. So the metric is defined on all pairs of vertices. So every edge exists with some value.

So that also follows with the triangle inequality. So if an edge does not exist, just make it the sum of the-- so if xz is not an edge, just make it the sum of xy plus yz. So in any case, you construct the initial path just by going down the tree and doing a naive DFS traversal. And, then, you correct that path by skipping over the duplicates.

So, finally, we'll end up with [INAUDIBLE] skipping all the duplicates we'll get a 1, 2, 3, 4, 5, 6, 1. And that's a valid cycle. So let's call this minimum spanning tree T. So that's your MST. And you are removing duplicate edges, and getting a cycle, C.

So now, actually, let's take another step back. OK, let's define our cycle, first. So let's call this cycle, this guy is C. And then, you're going from C, and you're deleting the duplicates, and you're getting C dash. So now what you have is cost of C dash. That's a [INAUDIBLE], but anyway.

The cost of C dash is less than equal to cost of C. And what is the cost of C? If you know the weight of your minimum spanning tree, what is the cost of C? C is just doing a [INAUDIBLE] of this traversal. So what is the cost of C, if you know your minimum spanning tree?

STUDENT: [INAUDIBLE]

AMARTYA SHANKHA BISWAS: No. So you're traversing every edge in the minimum spanning tree twice. So you're doing a DFS traversal, right? So you're going down, and you're coming back up. So you're going down, coming back up, backtracking, coming back down. So every edge is traversed, right? So it's exactly twice T.

So let's bring up a different board. So you know that cost of C is twice the cost of T. Does that make sense, why the traversal implies that you have every edge being visited twice, right? OK.

So, now, our claim is that the C dash cycle is a 2 approximation of the actual cycle. So why is that? So we've already [INAUDIBLE], so this also implies that C of C dash is less than equal to 2 of C T. Realize that this is not a valid cycle, but this is a valid cycle.

So, now, we need to show that this is somehow bounded by a star G. So how do you do that? Well, look at H star G. What is H star G? H star G is just a cycle, which goes through the optimal cycle, which goes through all the vertices and comes back to the parent vertex.

So this is H star of G. This is the optimal thing. Now, you can take an edge, e, here, and delete it. And then you'll get a spanning tree, because this is your optimal cycle. Remove one edge, and you get a spanning tree. So let's call that T dash.

Does that make sense, why that is a spanning tree? Because you had a cycle, and you remove one edge, so it touches all the vertices, and it's a tree. So it's a spanning tree, but it's not the minimum spanning tree. So you know that H star G, the cost of H star of G, is greater than equal to the cost of H star of G minus the edge we removed, is greater than equal to the cost of T. Make sense?

So you remove one edge, and then that is still greater than the minimum spanning tree. So, now, combining this guy and this guy, you get cost of C dash is less than equal to 2 [INAUDIBLE]. We know that cost of C is less than cost of H of G, so you get a 2-approximation. So does that make sense?

So that was a 2-approximation. That was pretty straightforward, We just constructed a spanning tree. You did a DFS traversal and removed duplicates, and you have a nice path. OK, let's just keep it down.

But here's the thing. It seems kind of wasteful to go through all the edges when you don't need to. So you're traveling down the tree, you're going back up, and you're traversing every edge twice. So it seems kind of ridiculous that you would be doing every edge twice. So what could you do better?

Well, before we introduce that, let's prove a couple of lemmas. So, first, we started with this. So let's say S is a subset of V. So you have a graph, and you make a subgraph. So you pick out some vertices.

So you pick out this one, and this one, and this one, and this one, and that is your S. And, so, you get a new graph which contains just those vertices. And, whatever, I just connect them. So the claim is that that graph also has a Hamiltonian cycle, the minimum cost Hamiltonian cycle, which is also the traveling salesman solution.

So let's call that H star of S. So it's some cycle, which looks like this, I guess, here. So H star of S. Now, the claim is that the cost of H star of S is less than equal to the cost of H star of G.

That should make intuitive sense, because you're taking only some of the vertices and trying to traverse them, and, in this case, you'll try to traverse all the vertices. However, this is only true because of the triangle inequality, and let's see why that is the case.

So, proof by contradiction. Say cost of H star of S is actually greater than cost of H star of G. OK, so, look at H star of G. It's a cycle, right? So you have something. Your cycle.

And, in this cycle, you have all the vertices of S. So pick them out. And, now, you have a cycle which has cost less than the optimal cycle in S, but it contains all the vertices in S. So what you can do is, now you can use the skipping lemma from before.

So, last time, we didn't move duplicate vertices. But, this time, what you'll do is, instead, you'll just skip over this vertex. We'll skip over this vertex. And so, every vertex that's not an S, skip over it. And that can only decrease the cost.

So, now, you've constructed another cycle, which contains only the vertices of S. So it's a Hamiltonian cycle for S, but it has cost less than equal to H star of G, which means that this can never be true.

So the important fact is, there, that if you have a subset of vertices making the restricted graph, the cost of the minimum Hamiltonian cycle is always less than equal to the one in the original graph. So, intuitively, that should make sense.

OK, next thing is something which might seem unfamiliar right now, perfect matchings. So you've seen perfect matchings in the content of bipartite graphs, right? So you find the minimum cost perfect matching. You do this flow thing, you send all the flow in, and then you connect the vertices with the capacities, and whatever.

So, it turns out, in a complete graph, you can still do perfect matching. So perfect matching is, you have a bunch of-- so, let's say, you need to have an even number of vertices, right, until we have perfect matching.

So, let's say, you have these varieties. So this is a perfect matching. So every vertex has one edge coming out of it, exactly one edge coming out of it. And it needs to be even, because, otherwise, that doesn't work out.

So that's perfect matching, and the minimum cost perfect matching is the minimum among all such things. And you did this for bivariate graphs. It's finite [INAUDIBLE] networks. So I'm not going to go into the algorithm for this. It's kind of complicated, but it uses linear programming, and you can find this for a complete graph, and it's polynomial, so that's good.

So, given a complete graph, you can find the perfect matching. [INAUDIBLE] for now. OK, one last thing we want to introduce is Euler circuits. So who has heard of Euler circuits before? Anyone? Sort of? OK.

So the reason we are-- so, OK, let's go back to what we did before. We had a tree, and the best way we found to traverse it was just going down, and going back up, and going down, and terribly messy.

So what we would, rather, want to do, is sort of traverse the thing without repeating edges. So, I don't know, you've probably seen this puzzle before. So you have this graph given to you, and the task is to draw this graph without lifting your pen off the paper.

So first, for example, in this graph, you could start here, go here, go here, and come back, and something, and there you got. So you can do that. But if you add on another lobe, here, then you can't make a circuit. You can still make a path, I believe. If you add another one, you can't even make a path.

So how does this work? Let's see. So, let's say, forget about the graph, for now. Let's say you're just drawing. So you start somewhere. You go to a vertex. You go to another vertex. Come back, go to this vertex, leave it.

And so, observe that, whenever you're making this drawing, you go to a vertex, and you leave it. Every time you hit a vertex, you leave it. Since there's the circuit, you just loop around, and every time you enter a vertex, you'll have to leave it.

What that means is that, even though this is not directed, if you drew out the actual path, you would see, the number of edges going into a vertex is equal the number of ones leaving, which means that every degree has to be even.

So if you go and look at this graph, which is that of lobes, this degree is not even. Neither is this, neither is this, neither is this. They're all 5, I think, yeah. They're all 5, which means that this can never have an Euler circuit.

So a graph can only have an Euler circuit if it has even degrees for every vertex. And the other way is also true. If a graph has even degrees on every vertex, then it must have an Euler circuit. That's not hard to prove, but there's a constructive algorithm you can use.

So, let's say, you're given this graph, for instance. You would simply go to the graph, just start at some random node, and then go through, and keep following edges until you can no longer following edges. So, let's say, you stop here. Then, you pick another edge, and start, and so on.

And, then, you can splice these cycles together at some point. So it's kind of a [INAUDIBLE] argument, but it should be sort of intuitive why you can construct another path, given an even degree. You just perform searches. You just create cycles, and you splice them together.

But, for now, just take it as fact that a graph is an Euler circuit, as in you can draw it without lifting your pen off, if, and only if, every vertex has even degree, so why is that interesting?

So, let's say, we could add some edges to our tree and turn it into one of those nice graphs. Right now, this is not, right? This is degree 1, degree 1, degree 3, degree 1, degree 1, degree 3. All of them are odd, so that's not good.

But, let's say, you could add some edges in, turn it into an Euler circuit, and then you could do a nice reversal off it, and, maybe, that will give you a better approximation. So with that hope, let's look at the algorithm.

So what you do is, you go back to your tree. Let's just leave it at that. So, now, let's see. So this is degree 2, that's good. This is degree 3, that's not good. Degree 1, this is fine. This is degree 1.

This also is degree 1. This is degree 3. Actually, let's get rid of this, so it does not have degree 3. That's a lot of vertices. OK, there we go. So, in this graph, you see that you have 1, 2, 3, 4, 5 vertices which have degree odd. So, now, we would like to add some edges to turn this into an Euler circuitable graph. So how do we do that?

So let's call the set of odd edges S. Odd vertices, sorry. So you take the set of odd-degree vertices. Now, go back to perfect matchings.

So what does a perfect matching do? It adds edges to that graph so that everything gets degree increased by one. So if you increase the degree of all the odd vertices by 1, everything turns even, right? So you take the set of odd vertices.

OK, another thing to observe. Realize that, how many odd vertices can you have? Can you have an odd number of odd vertices? Because that would screw up the whole thing where we needed an even number of things.

So why is that not possible? Why can you not have an odd number of odd vertices? So the thing is, that, let's say, you have some graph, and what is the sum of the degrees of the graph? Let's move to a different board for this.

So you have a graph, G, and you want sum of degrees. So di is the degree, for all V. So what is this equal to? So, let's say, you have this graph. So, now, if you count the degrees of every vertex, you're basically counting the number of edges coming out, which means that every edge is counted twice, once for this end point and once for this end point.

So this edge is counted twice, for here and here. This edge is also counted twice. So, basically, the sum of the degrees is nothing but 2 times mod of E. Does that make sense? So this is even.

Now, let's say you take only the odd vertices out. So the even vertices are good. This is good.

This is good. This is not good. This is also not good. So these are the odd vertices.

So the sum of the degrees of the even vertices are, by definition, even. So, if you remove them, the sum of the odd vertices should also remain even. And so, you have the sum of odd degrees becoming even, which means that you'll need to have an even number of them. Make sense?

So, going back to this, so there's a lot of branching off, here. But going back to the main point, here, is that you have an even number of odd vertices. So consider the restriction of the original graph, G to this set, S. So, now, we're going to the first thing we did, there, where we considered a restriction of the graph to a certain set of vertices.

So we take the set, so 1, 2, 3, 4, 5. So you take these five vertices. And you consider the graph that is restricted to these five vertices. Oh, sorry, there should be six. Oh, that's interesting. Oh, there we go. That's the other one. So those are the six vertices which have odd degrees.

Now, you find the perfect matching with your polynomial-time algorithm, and you get something. So you can now add those edges back in here. So these are your new edges, and, let's say, something, this one.

OK, so you get three new edges. And, now, realize that all the degrees are now even, so now you can do your Euler circuit. Also, let's call this matching M. So this is a set of edges, M. Let's call the original tree T, and the new thing that is formed, is T union M.

So you're taking all the edges from T, and you're adding all the edges from M. So, realize that you can have multiple edges, but that's fine, because Euler circuits allow you to have multiple edges. So, now, you take this graph, and you find this Euler circuit.

So that is basically this thing. So this set of edges in some order, and that order exists. So the cost of the Euler circuit-- let's call it C-- is equal to the cost of T plus the cost of M, right? Because you're traversing all the edges in your graph. So that is the cost for the Euler circuit.

Now, my claim is that-- so this is the difference between all the nodes in the graph. And, remember, you can do the whole duplication argument, from before. So, where's that? Oh, there.

So you can do a duplication argument, so you're visiting all the edges in the graph, all the vertices in the graph. You remove the duplicates, and you have a valid path. Now, let's see if this valid path is actually a better approximation.

So, again, you will go from C. So this C, you will go to C dash. And this will give you, as before, cost of C dash is less than equal to the cost of C, So, now, you're only interested in cost of C. So what are we doing there?

So we know that the cost of C is equal to cost of T plus the cost of M. So, from the previous problem, we know that the cost of T is less than equal to the cost of H star of G, right? So this is less than equal to cost of H star G.

So this guy is less than the optimal Hamiltonian path. What about this guy? So let's look at the actual H star G, again. So, actually, let's look at H star S.

So remember that S is the set of odd vertices, and that is where this matching is done. So H star S is nothing but the optimal Hamiltonian cycle on that restricted graph. From our previous lemma, which is down here, we know that H star of S is less than H star of G.

So we know that cost of is less than equal to cost of whatever, G. So, now, let's just look at H star of S. Now, we construct a matching. We take every other edge. We take this one, and we take this one, and we take this one. And look at the alternate set, also. So take this one, and this one, and this one.

So look at the blue set and the red set. Since it's part of a Hamiltonian cycle, they're both matchings, right? So let's call the red one M1, and the blue one M2. So I'm not saying they're perfect matchings. They're not the minimum matchings, but they're definitely perfect matchings.

And because we know the perfect matching is M, or the minimum matching is M. Cost of M is less than equal to cost of M1, and, also, the cost of M is less than equal to the cost of M2. That make sense? Because M was the minimum cost matching, you have cost of M is less than any other matching, which is constructed from this.

Again, so this implies that the cost of M is less than equal to half cost of-- OK, let's not write this here. Let's get a different board.

So cost of n is less than equal to half cost of M1 plus cost of M2. Since both of these guys are larger than this, their average is also larger than this, right? But what is cost of M1 cost of M2?

This is nothing but half cost of H star of S, right? Because the Hamiltonian cycle is constructed from M1 and M2.

And by our lemma, this is less than equal to half cost of H star of G. So, now, we have all we need. This is less than equal to half cost of H star of G. And then, add them together, you get cost of C dash is less than equal to cost of C, is less than equal to the sum, which is equal to 3/2 cost of H star of G.

So that's the proof. So there's a lot of things going on, here. Let's try to go back and see what we did. So we took a minimum spanning tree, and then we tried to remove all the odd degree vertices.

So we took all the odd degree vertices, and made a perfect matching, the minimum cost perfect matching. So we added edges just to make everything even degree. Then, we took the Euler circuit on that graph, and we removed duplicates. So that's fine, that's the easy part.

But you do [INAUDIBLE] circuit on that graph, then you argued that, because all the circuits in that graph is just the sum of the edges in the spanning tree plus the sum of the edges in the matching, which you added later.

So the spanning tree had already bounded, in the previous argument. The matching was bounded by taking the optimal Hamiltonian cycle, decomposing it into two matchings, arguing that those two matchings are not the optimal matchings-- not necessarily the optimal matchings, but they're less than equal to the optimal. So they're even, there.

And, so, the cost of the optimal matching, which you were using in our constructive path, constructive Hamiltonian cycle, is less than equal to both of these matching. You add them up, you get that bound, and it works.

So questions on any of the steps? This is a lot of branching off and then coming back together. Yeah.

STUDENT: The last one's 3/2, right?

AMARTYA SHANKHA BISWAS: Yes. That is not 3 over 3. That would be p equal to np. Anything else? You're free to go, or ask questions, or whatever.