PROFESSOR: Today we start a-- we're going to take a little break from parametrized complexity and talk about something called exponential time hypothesis, which I've mentioned a few times but we haven't really talked about its many cool consequences. I'll abbreviate it ETH. And the idea is think about 3SAT. and the claim is it has no 2 the little of n time algorithm.

So of course you could make analogous assumptions about KSAT set for various K. But in general, the hypothesis is that for any fixed K, there is no 2 to the little l of n time algorithm. So 3SAT is the smallest one where that should hold. 2SAT, of course, is easy. This is not the maximization version.

And our evidence for this is merely that we haven't been able to find such an algorithm. There are better than 2 to the n algorithms. The obvious algorithm is 2 to the n, try all possible assignments times n. But we won't worry about the non-exponential part. The best algorithm to date is 1.31 to the n.

That's just from three years ago. So it's an active area of research. Many people have tried to find better SAT algorithms. But conjecture is you cannot get, like, 2 the square root of n, the sum, and be complete problems have 2 to the square root of n time algorithms-- we'll see some today. I'll mention some. But seems like 2 to some constant times n or some constant to the n power equivalently is the right answer for 3SAT.

Now one issue to talk about very briefly-- it won't actually matter-- is what is n? Normally n is the entire problem size, which would be the number of clauses, essentially, because each clause has three variables. But the usual assumption is to say this is the number of variables and m is the number of clauses. But these turn out to be the same thing.

So one assumption is equivalent to the other assumption, so henceforth we don't need to worry about this. In general, m is at most n cubed, because each clause can have three different variables, so there's only n cubed possible clauses. But when we're worried about this little o thing, we actually care about polynomials, because we're in the exponent.

So it turns out you can sort of think of m and n as being linearly related-- that's OK-- by something called sparsification lemma. It says that if you have a formula with a superlinear number of clauses, you can convert it into some vaguely reasonable number of formulas with a linear number of clauses. It's, like, 2 to the epsilon n different formulas, each with a linear number of clauses.

And so the end effect we need to know is that these two are equivalent. We can think of situations where clauses and variables are linearly related. So that's the hypothesis. Of course, we don't know how to prove it. If we proved it, it would imply p does not equal np. This is sort of a strong form of p does not equal np from the SAT perspective.

Because we have so many reductions from SAT, you can turn this lower bound. If you assume this lower bound about SAT, you can prove corresponding lower bounds about other problems we've seen. While I'm here, let me just tell you another version of ETH called strong ETH, which is that CNF SAT has no 2 minus epsilon to the n algorithm.

So the idea is there's some ideal constant to the n for each KSAT. But as K goes to infinity, that constant goes to 2. So if you have really large clauses, you can't be 2 to the n, roughly. This is for all epsilon greater than 0. So I won't talk about anything that needs this assumption, but there is some work on-- you can get very specific bounds on what the constant should be if you assume strong ETH.

Let's take a review of past reductions. And I'm going to start with 3-coloring a graph for here. So this was our proof that 3-coloring was hard. There was a clause gadget, variable gadget, this thing of constant size.

And the point is, if I start with some 3SAT-- this is a reduction from 3SAT-- if I start with a 3SAT instance that has n variables and m causes, the number of vertices and edges over here will be some constant times n plus m. So 3-coloring, we get order n plus m vertices and edges.

So what does that give me in terms of a lower bound? Well, if I assume ETH that there's no 2 to the little of n and no 2 to the little of m time algorithm for 3SAT, then I get a corresponding thing that there's no 2 to the little of n time algorithm for graphs with number of vertices and number of edges linear in n. So I state it this way because for graphs again, there's the issue of sparse versus dense graphs. I could just say for sparse graphs here, which each have a linear number of edges.

So this is, I should say, ETH implies this. So while I thoroughly believe p does not equal np, p does not equal np, ETH is a little bit more unknown. So I'm going to explicitly mention ETH teach every time I use it.

And more generally if we look at various past reductions, all we need to measure is something called size blowup. Remember an np reduction Karp style, you start with some instance x, you make an instance x prime. And the size of x prime-- this was our reduction f-- the size of x prime should be polynomial in the size of x.

What's that polynomial? That's your blowup. So if we have n, which is the size of x over here and n prime, which is the size of x prime, this was polynomial in n and that is the size blowup.

And so in particular, if we have linear blowup, like in this example of 3-coloring, then we preserve the ETH statement. No 2 to the little o of n time algorithm, meaning if there was no 2 to the little o of n algorithm for problem A, then there'll be no 2 to the little o of n algorithm for problem B, because you could just convert to only get a linear size blowup, run that algorithm if there was a 2 to the little o of n algorithm. And then that solves A in 2 to the little o of n time. So the usual argument, but now we're giving explicit bounds on the running time.

More generally, if you don't have linear blowup, let's call this function b of n. Let's say size of x prime is always at most some function b of n, b for blowup. Then if there's no 2 to the little o of n algorithm for A, then there will be no 2 to the little o of b inverse of n algorithm for B.

So for example, if you have quadratic blowup, b of n is n squared, then you will say that there's no 2 to the square root of n, no 2 to the little o of square root of n algorithm. You can imagine how that goes. So here are some nice examples of reductions we've seen. This one was from lecture seven.

Then from lecture 10, we had dominating set-- oh, sorry. I have one more before dominating set. This was in the context of proving planar vertex cover was hard. First, planar 3SAT was hard and then a reduction from 3SAT. But ignore the planar issue, because things are going to be a little different for planar graphs. But for general graphs, this was a reduction from 3SAT.

And there was a constant number of vertices and edges for each clause. And then also here, we had to make a copy of variable for each occurrence, but the total number of recurrences of all variables is linear, because there's only three occurrences per clause.

So the total number of vertices in these variable gadgets is also linear, so the whole thing is linear. Size blowup is linear. And so vertex cover is another example of this type of result. Assuming ETH, there is no 2 to the little o of an algorithm for graphs whose number vertices and edges is order m.

AUDIENCE: Is there a name for that class?

PROFESSOR: That should have a class, but-- I can make one up, if you want. We could call it an ETH-style graph problem, say sparse graphs no 2 to the little o of n assuming ETH. So it's sort of saying it's linearly related to SAT, but as far as I know it doesn't have a proper name.

It should, because I had to write it many times in my notes. Another one was dominating set. I don't have a slide for this, because there never was one, because the reduction was so simple. It was if you have an edge in the vertex cover instance, you convert it into a subdivided edge and the original edge for dominating set.

And then you have the same problem. It's the domination. So it's the same thing as vertex cover. You never want to put this thing in the dominating set. You might as well move to one of the neighbors. So that's again, a linear size blowup. And so dominating set is also in this class, who shall remain nameless.

And another one is Hamiltonicity. We saw a couple of different proofs. This one is from lecture seven. This was ostensibly for directed, thought it also claims to work for undirected graphs. Linear? Maybe I'll jump to the next one, because it's a bit of a stronger result.

This was maximum degree 3, Hamiltonicity directed. This was lecture eight. It's also linear. I mean, the main diagram is this. It's linear if you're not aiming for planar graphs. And then there's no crossover gadget here. And so the total complexity of all these things is linear.

So that's cool. So Hamiltonicity is another example of a graph problem with no 2 to the little o of n algorithm, assuming ETH. And from this proof-- this was for directed max degree 3-- and it was also bipartite. And then we reduced that to undirected-- sorry.

We reduced from planar directed max degree 3 to planar bipartite undirected max degree 3. And that's of course also linear. So all these things we get for free. We did them in different contexts. That was for planar graphs. This one was for APX hardness for independent set.

But we use the same proof. And we have this biclique for the variable. Now here you start to worry this could be quadratic blowup. But this was actually a reduction from 3SAT-3. So that we didn't have a very large clique there. It was actually only three nodes in it, so it's more like two edges.

But in general, 3SAT, any constant would suffice for that proof. You also have to check that this reduction from 3SAT to 3SAT-3 is OK. But it's, again, linear blowup because the total number of occurrences of all variables is linear. So that was a whole bunch of free results-- independent set, 3SAT-3.

That's not a graph problem, so I won't write it in this list. Now, normally independent set is the same thing as clique. In this universe, that's not quite right because we're talking about sparse graphs. For clique, it's still the case that there's no 2 to the little o of number of vertices algorithm. But the number of edges used to be linear for independent set and becomes quadratic for clique. So you have to be a little careful with clique.

So all is good as long as we're talking about non-planar graphs. What about planar graphs? Well, this is not true for planar graphs. In general, you tend to get 2 to the square root of n algorithms, and that's tight, assuming ETH.

So for example, planar 3SAT we had a crossover gadget. And in the worst case, there are a quadratic number of crossings. And so the blowup in our problem size, because we spend some constant number of vertices per crossover, the blowup is quadratic.

And so for, say, planar 3SAT, ETH implies no 2 to the little o of n or 2 to the little o of m algorithm-- sorry, with square root. So with 3SAT, it's a little annoying because we have to think about variables and clauses separately.

So the size blowup is not quite as uniquely defined. But just analyzing number of variables, number of clauses separately, the blowup is quadratic in both. So that's the lower bound we get. And then I have planar 3-coloring, vertex cover, dominating set, Hamiltonicity, independent set. All of them have the property that ETH implies. No 2 to the little o of square root of n algorithm for planar graphs.

Now planar graphs are always sparse, so I don't need to worry about how many edges versus vertices. n is within a constant of both. How you prove that? Exactly the same proofs that we just looked at.

They were all actually proofs for the planar problem, but they all had some kind of crossover. Either they started from planar 3SAT, in which case they were already quadratic. Like this one was from planar 3SAT and it was a linear blowup from that. So it's only quadratic overall. This one was again, from planar 3SAT and linear after that.

This one was from 3SAT. And then there was a custom crossover gadget, which I don't have the slide for here. But for each of these crossovers, we paid something, so we get quadratic from 3SAT. And that's linear, of course. And this is not a planar independent set reduction, so I don't have one here.

You have to fill in your own. And one other one was coloring. We did the planar 3-coloring gadget. Again, you pay constant for each crossing. So quadratic reduction from 3SAT-- all of them end up being quadratic overall. Independent set's the only one I haven't shown you. And-- cool. So this is a sense in which even though the planar problems are np hard, they a little bit easier. Question?

AUDIENCE: So you mentioned that was [INAUDIBLE].

PROFESSOR: Yeah. So I think-- I should double check. I'm pretty sure all of these problems have 2 to the square root of n time algorithms. I'm confident enough that I will write it down. I think the general approach is Lipton Tarjan separator.

But that's about the level of detail I remember. Oh, yeah-- also, all planar graphs have tree width order square root of n. And generally, that will give you such an algorithm.

So that was-- question?

AUDIENCE: Are there any of these problems that you can, in a sense, preserve the difficulty in a planar graph?

PROFESSOR: Yeah, that's a good question. We might get to some. I'm about to shift gears into parametrized complexity. And in that setting-- I would say generally no.

But there are certainly some exceptions where you can encode a non-planar problem into a planar structure. But most natural problems tend to be like this. But there definitely are examples. We might even see one.

This is sort of-- this could have been in lecture two, and maybe it should have been. But ETH is nice because it gives you a bit more of a quantitative sense of how much running time you should expect out of your algorithms. It gives you motivation for going for linear blowup when possible, or at least minimizing your blowup and lets you distinguish between planar and non-planar problems.

But we're in the middle of parametrized complexity. And I mentioned all this in particular because it has an even bigger impact on parametrized complexity. So let's shift over to that world.

Now first of all, we get two sort of trivial consequences just from these statements. They're trivial, but in some cases they're actually interesting. So they're easy to prove, but actually give tight answers for a few problems.

So for the natural parametrizations, a vertex cover is a vertex cover size at most k. Longest path, which is the optimization version of Hamiltonicity, is their path-- in the parametrized version, is their path of length at least k? Dominating set of size k, independent set of size k upper bound and lower bound.

In particular, there can't be a 2 to the little of k times polynomial in n algorithm, because there's no 2 to the little o of n algorithm. This is assuming ETH. Because in particular, kb could be n. Now this is not exactly what we care about. What we care about is whether there's an f of k times polynomial in n algorithm.

But this at least gives you a lower bound on the f. So in particular, for dominating set and independent set, this is not a very interesting result, because in fact we will show, assuming ETH, these do not have FPT algorithms at all. There's nothing of that form. But for vertex cover, it's interesting because that is FPT.

And there is a constant to the k times polynomial n in n algorithm. We saw a 2 to the n times n algorithm. And this shows that that's tight. So there's no better-- it gives you a bound on f. For vertex cover, c to the k is the right answer for some constant

c. And if you assume strong ETH, you can actually figure out what the-- well, you could try to prove with the constant is.

We don't know the right answer for vertex cover. Some of these problems, we do. Longest path, same deal. It's FPT, so it's easy to find short paths. And the algorithm is like 2 to the order k times polynomial in n.

And similarly for planar problems, if we have ETH, there's no 2 to the little o of square root of k times polynomial n for those same problems on planar graphs.

For clique, actually this should also work. Clique is OK because k is the number of vertices in the clique. And so even though the number of edges is quadratic, this would still hold. For a planar clique, of course it's polynomial. So I can't put clique down here. The maximum clique size is 4, so there's an n to the 4 algorithm.

Again, this is interesting because for dominating set, independent set, vertex cover and longest path, there are 2 to the square root of k times polynomial in n algorithms. So this is actually a tightness result. There exists 2 to the order square root of k n to the order 1 algorithms for planar graphs for those problems.

This is called subexponential fixed parameter tractability. And there were a bunch of those results in the early 2000s. And then a theory called bidimensionality kind of characterizes when it's possible, or gives you a big set of examples where it is possible. But that's algorithm, so we're not covering it.

So all well and good. So for planar or dominating set, that's interesting. But for general dominating set, we know dominating set is w2 complete, we think that means there's FPT algorithm. Independent set in clique, our w1 complete, we also think that means no FPT algorithm. Assuming ETH, we can actually prove that. So let's say there's no FPT algorithm for clique/independent set assuming ETH.

So that's a theorem we will prove. If you believe in ETH, then w1-- these problems are complete for w1-- w1 does not equal FPT. These are the FPT problems. And in fact, we can prove a much stronger bound.

Very non-FPT-- these algorithms generally have an n to the order k algorithm, or if they're in xp, then they have some n to the k to some constant algorithm. But we can't even reduce that exponent below k for any-- for clique and independent set, let's say.

And if you reduce clique and independent set to some other problem, you can, just like we've been doing over here, you can keep track of the parameter blowup. And if it's a quadratic blowup, then you'd get that there's no n to the square root of k algorithm. We'll actually do that in a moment for planar graph problems. But for general graphs, clique and independent set, no f of k for any computable function f times n to the little of k algorithm. So this is much stronger than FPT does not equal w1. And this is a result from 2006, so fairly recent by Chen et al. So let's prove it. It is essentially a reduction from 3-coloring. But it's unlike most reductions we think about, because-- well, it's unlike parametrized reductions. Question?

AUDIENCE: Sorry, so is this [INAUDIBLE] from claiming that the xp hard these problems?

PROFESSOR: Yeah. xp hard is really way up there. None of the problems we've talked about are xp hard, unless something happens with p versus np.

AUDIENCE: But xp [INAUDIBLE] problems that you have--

PROFESSOR: These are in xp, but they're also in-- these problems are actually in w1, which is much smaller than xp. Yeah, I mentioned xp because of the n to the k to some constant is related in the same vicinity. But it's not directly about xp.

So normally when we do parametrized reductions, we start from a parametrized problem and we reduce to a parametrized problem. Here, we are reducing from an unparametrized problem. 3-coloring has no parameter. And we are going to reduce to clique, which has a parameter, namely the size of the clique.

So it's a little weird, but you've got to get started somehow. So we're going to introduce a quantity k and set it how we want to. So here's the idea. We are given an instance of 3-coloring. We're given a graph. We're going to split the vertices into k groups, each of n over k vertices. And remember, we know that 3-coloring has no

2 to the little of n time algorithm. That's what I just erased, assuming ETH.

So I'm going to choose k in a little bit. But let me first tell you the reduction. So we're going to create a new graph. Let's call it g prime, with k groups of not n over k, but 3 to the n over k vertices.

Why 3? Because we are going to think about all possible 3-colorings of those n over k vertices. So it's corresponding. For every group up here, we're going to just write down every possible 3-coloring. So obviously, n over k has to be quite small, every possible 3-coloring of those and n over k vertices.

So the intent is that in our clique problem, that we want to choose exactly one vertex from each of these groups. So k is supposed to be the size of our clique. That's why I wrote it this way.

So at the moment, I have vertices but I have no edges. Each of the groups is going to be an independent set, so that means you can only choose at most one vertex from each group, to make a clique.

And we are going to connect to colorings if they're compatible. by an edge nG prime if compatible. So the idea is, here is one group, size n over k. Here is another group of size n over k. And if you color these vertices some colors-- I'm only using one color-- and you color some colors over here, now it's coloring within the group, but there are some cross edges between here which may be incorrect. They may be monochromatic.

And so we check whether the coloring of this and the coloring of this is consistent with all the cross edges between those two groups. If it is compatible, if it's a valid coloring of both groups, we connect them by an edge. This coloring corresponds to single vertex in G prime and this coloring corresponds to a single vertex in G prime.

And we add an edge if it's OK. We don't add the edge it's not OK. And if we're looking for a clique, that means we need to choose a coloring for each of the groups where everything is pairwise compatible. And that represents all the edges. Every edge is either within a group, in which case it was taken care of at this stage.

I guess I should say is at most, 3 to the n over k. I only want valid 3-colorings. Or the edge crosses between two groups and then it will be considered when we think about whether there's an edge. In a clique, there are pairwise edges and so everything is pairwise compatible.

So never mind the claim. You should be convinced this is a valid reduction, in terms of a correctness standpoint, from 3-coloring to k clique for any k. The construction depends on k.

So what do we set k to? Here is a setting for k that will work. I don't have a ton of intuition for this, other than the algebra works. Essentially, we need to set it just-- we want to set k to, like, a tiny, super-constant thing. So just a little bit, little omega of one.

And I need to give that little of k a name. So I'm going to say, let's say k clique could be solved in f of k times n to the k over s of k time, where s of k is some monotone, increasing, and unbounded function.

I need that s goes to infinity. That's the meaning of little of k is that you can divide by something. And you can assume without loss of generality that the something is monotone increasing, but in particular it should go to infinity as k goes to infinity. It might go there very slowly. It could be, like, 1 over 2 to the k or something. But something little-- s is little omega of 1.

But now I have this quantity s. And I'm going to set k as large as possible so that f of k is at most n and k to the k over s of k is at most n. My goal here is to make k a function of n. And so one choice of k is basically f inverse of n.

So f, remember, was the dependence on k, so here. This will turn out to work. So you can think of k just being f inverse of n. But there's actually another constraint. It's another inverse thing. I want k to be at most, the inverse of this relation. So I'm basically taking the min of these two functions of n that will be a function of n, which is growing.

I mean, you can check. If you set k to be a constant, of course this is true. If you set k to be a constant, of course this is true. So you can set it a little bit superconstant by inverting this relation. That gives you some value of k that would satisfy this, some function k equals k of n that would satisfy this.

And I want to take the min of those two. Still a growing function of n. We'll need that in a moment. And I get these two inequalities.

Now it is just a computation of how much running time I have. So I want to plug this algorithm-- this was an algorithm for k clique. I have this instance of k clique, which looks a little weird because it's got potentially a lot of vertices.

I'm just going to plug that in. This is my n prime, the new-- well, the number of vertices is this times k, because they're k groups. So a number of vertices in G prime is k times 3 to n over k at most.

So we just plug that into this running time. And we get f of k times that number of vertices, k times 3 n over k at most. So less than or equal to the power k over s of k. And now we do some manipulation. We know that f of k is at most n. That will be enough for this term.

This is at most n times-- I'm going to split this apart. So we have k to the k over s of k power. And then separately, we have 3 to the n over k to the k over s of k. Again, k to the k over s of k, that's something I get to assume is less than or equal to n. So this is less than or equal to n squared.

And then the k's cancel. And we're left with 3 to the n over s of k. I'm going to remind you k is a function of n that is unbounded and monotone increasing. So this is 3 to the little o of n, also known as 2 to the little o of n.

So I just needed to choose k to be slightly superconstant. And I wanted to get rid of these terms, so I made them at most n and took those inverses, took them in. And boom, we get a contradiction. This contradicts ETH. This implies ETH is false.

So if you assume ETH, running backwards you get that k clique cannot be solved in such a running time. And so we get this very strong lower bound. There's no f of k times n to the little o of k algorithm for k clique. So in particular, k clique is not fixed parameter tractable. I think that's pretty neat.

And henceforth, you care about parameter blowup. I mentioned it briefly last class. But in general, you map some problem x with parameter k into a new instance x prime with parameter k prime. And k prime just has to be bounded by any function, any computable function of k.

But if it's a linear function, you preserve this strong bound. If it's quadratic function, then you get there's no f of k times n to the little o of square root of k. If it's an exponential function, which is fair game here, you get a weaker bound. You still get that there's no FPT algorithm. But you don't get a nice-- not a very impressive bound in terms of n on the right.

AUDIENCE: Is there a name for this type of reduction?

PROFESSOR: I don't have one. There's only one that I know of, so I don't think I will try to come up with a name. Once you have this, you can-- last class we reduced k clique to all sorts of things. And so we get a lot of-- for now, you just reduce from clique or variations. And so you get lots of good stuff. What do you get?

Last time we covered a reduction from clique to multicolored clique and independent set. And if you look at that proof, k prime equals k. We had a quadratic blowup in the problem size, but the parameter didn't change at all.

So this is good news. That means this problem, or these two problems, has no-- assuming ETH, there's no f of k times n to the little o of k algorithm. And also, we covered a reduction to dominating set. Even though dominating set was w2 hard, we still reduced from multicolored clique to dominating set.

And then from dominates set, we could reduce to set cover. All of these reductions preserve k exactly. All we need is that they preserve it linearly. But then we get this kind of result for all of those problems.

We covered a reduction for partial vertex cover. But I think the reduction we covered was not linear. But there is one. So I'll just state this as a result, but this is another one where we covered a reduction, but it wasn't the most efficient one. I think we lost a quadratic amount. Any questions? Yeah?

AUDIENCE: Do we happen to know if FPT and w1 are separated assuming only [INAUDIBLE] and not assuming ETH?

PROFESSOR: We do not know that. The best classical assumption is ETH implies w1 does not equal FPT. I also don't know offhand whether FPT does not equal w1 implies p does not equal NP. I think there's this result along those lines, but I'm not sure if that's literally true. So intuitively it's stronger, but-- other questions?

AUDIENCE: So this is strictly better lower bound than those over there?

PROFESSOR: Right. Good question. So before we switched to parametrized land, we said-- like over here, we had there was no 2 to the little of n algorithm. Here we're getting that there's no f of k times n to the little of k algorithm.

I think that is stronger than the old bound. Though I guess you have to think about it problem by problem. It depends on how k could relate to n in general. I think for these problems though, it is a stronger result.

Because k is at most n. And k can be close to n. So the next topic-- just taking a little breather. This is all good for non-planar graphs. For planar graphs, we actually haven't seen any w1 hardness results yet. And that's because a lot of planar problems are FPT.

There are, in fact, 2 to the square root of k times polynomial in n algorithms for a ton of planar graph problems. But there are some that are hard, some that are w1 hard. And as you might expect, this k becomes the square root of k because we get a quadratic blowup, for some problems-- not quite all of them.

So this comes back to Jason's question. And maybe I'll go up here. Let me briefly mention in general if k prime of x prime is at most some g of k of x, this was part of our-- in the definition of parametrized reduction, then if there's no f of k n to the little of k algorithm for problem A, then there is no f prime of k prime times n to the little o of g inverse of k prime algorithm for B.

So I think the analogous statement was up here for size blowup and regular np reductions. But for parametrized reductions, I mean, the dependence on k is just an arbitrary computable function. So that doesn't change. But the exponent changes correspondingly. So if you square k, we get the square root in the exponent.

So let's do some planar problems. I'm pretty sure all of the w1 hardness results for planar problems are within the last five years. So it's a pretty recent direction. And the key insight is a problem called grid tiling, which I have a slide for. So this is a problem invented by Daniel Marx in 2007.

And so the input looks like this and the output looks like that. So in general, you're given a k by k matrix of sets. Each set has some number of 2D coordinates. These coordinates range between 1 and n.

So it's k by k, small matrix, but each cell has a ton of stuff in it, up to n squared pairs. And your goal is to-- in this example, all the numbers are between 1 and 5. So n equals 5. And it's 3 by 3, so k equals 3.

Your goal is to choose exactly one element from these sets, such that if you look in any column the first coordinates are all the same. Here, the first coordinate are all 1. Here, first coordinates are all 2. And in any row, the second coordinate is the same. Here they're all 4. Here they're all 2. Here they're all 3.

So that's a valid solution. As you might expect, this is np complete. But furthermore, it's w1 heart. I should say sij-- just gives you some notation-- is a subset of 1 up to n squared for all i and j in 1 up to k. That's what the input looks like.

The squared means you have ordered pairs. And then your goal is to choose an xij out of each sij so that in any row, the first coordinates match any column. Sorry. In any row, the second coordinates match and in any column, the first coordinates match.

So claim is this is w1 hard. And also now, we know w1 hardness is not the most we could hope for. We also want to know what the parameter blowup is and how we can relate it back to ETH. So here we will get the same bound.

There's no f of k times n to the little o of k algorithm, assuming ETH. So here we will not lose a quadratic thing. But notice that this thing is k by k. So, while we've defined the parameter to be k, there's kind of a quadratic amount of stuff going on.

You're selecting k squared different things in the solution. That's still OK. I mean, k is still the number of rows or columns of the matrix. But typically, the reason this problem is interesting-- this is, like, a starting point for planar graph problems, because you can replace each of these cells of the matrix with the gadget that somehow represents all the things in that set, but that now these constraints that all the things in the column agree in the first coordinate, you can think of as a local constraint.

Because really, you just need that the guy you select here has the same first coordinate as the guy you select here. You only need to constrain adjacent cells. Because if his adjacent cells are equal, then the whole column will be equal.

And if adjacent rows have equal second coordinates, then the whole column will have equal second coordinates. So as long as you can build gadgets that just check with their neighbors, that will give you a kind of planar graph structure, or a 2D geometry structure if you're doing a geometric problem and it lets you work with things kind of locally.

But when you do that, of course, typically k becomes k squared. And that's where you get the square root of k up here. But it won't disappear yet. So how do we prove this? We're going to reduce from clique because that is our favorite w1 hard problem. And it has this kind of bound.

And so it's going to be a linear reduction. In fact, k prime will equal k and n prime will equal n. n here is the maximum coordinate values in the original problem. n prime is the number of vertices in the clique. And I'm going to write down the reduction and then show you a picture. It's hard to actually draw the full reduction. It's easier to write it down generically and then show you kind of a little slice of a real example.

It's a little bit confusing, because there are four parameters lying around. There's which cell are you in, which I'm denoting by ij. i is which row you're in. j is which column you're in. So this is the set of things at row i column j. But then separately, there the coordinates that are inside the cell.

And here I'm denoting that by vertices, because for us, what this says is that the vertices map 1 to 1 with coordinate values. But these coordinate values are different from the ij coordinate values. The ij's are always between 1 and k. These coordinate values are between 1 and m.

Probably should have a term for those, that distinction. But such as it is, ij is between 1 and k. v and w are between 1 and n. So there's two types of cells. There's cells on the diagonal of the matrix and cells off the diagonal. This is for i not equal to j.

On the diagonal, you just have pairs of the form vv. So that's supposed to represent the vertex. And so basically what you choose on the diagonal is going to correspond to the clique that you want. Because the diagonal has size k, and so each diagonal item is going to have to choose a vertex. It's going to turn out that vertex will be in a clique.

Why will it be in a clique? Because the off diagonal entries are going to force that there are edges between corresponding vertices. So the off diagonal entries will have, for every edge-- and we're assuming no loops here-- for every edge, we put vw an item in the set of pairs.

And if this is an undirected graph, we'll put vwnwv. And so in fact, all of the off diagonal entries look the same, I guess. And all of the diagonal entries look the same in terms of the s sets.

So let's look at an example. Suppose you choose this 2-2 diagonal entry to be vertex i. Didn't assume very much. But from the constraints of the grid tiling problem, we know that the whole row here has the same second coordinate.

And the whole column here has the same first coordinate. So if you choose the vertex i here, that forces i to appear throughout there. And if you look at some other vertex, vj on the diagonal, then same thing happens. You have j second coordinate here and j first coordinate there. I see there's a slight typo on these slides. This should be a j. That should be an i. The colors are right, though. So just look at the colors.

Now, for this to be in the set, there must be an edge between vi and vj. And this is true for all i and j; therefore you have a clique. Now one thing that's important is that vi is distinct from vj.

Otherwise, you could just put vi in all the diagonal entries, and everything is vi vi. But because we said v does not equal w for these sets, the fact that there is a valid choice here, the fact that vi vj is a valid thing in this item means that vi does not equal vj.

So these are all distinct vertices. There's exactly k of them. And so this problem has a solution if and only if there was a k clique in the original graph. Clear? I guess these entries are technically correct if you view these as unordered pairs. Because we're in an undirected graph, everything is flippable.

So that proves that grid tiling is as hard as clique. And it was a linear reduction. We started with value k. We ended up with a thing whose parameter was k.

AUDIENCE: Say something again like, it just seems like you've just redefined your n and k to be square root of what you might normally.

PROFESSOR: Yeah, so if you prefer, you could define k to be the number of cells in the matrix. And then what you would get here is there's no f of k n to the little o of square root of k algorithm, assuming ETH. It's just a matter of what you define k to be. You're going to either lose a square here or later.

And I think-- so I'll show you why in a moment, why you might define it this way. Because here's going to be a planar graph problem, kind of, that does not blow up k at all. Turns out, sometimes you don't have to blow up-- this k turns out to be the correct k.

So let's do that example. It's called k outer planar list coloring. There's two things I need to define-- list coloring and outer planar.

Let's start with list coloring. So in list coloring, given a graph and for every vertex, you're given a list of valid colors.

And your goal is to color the graph. Again, no edge should be monochromatic. And the color you choose for vertex v must be on the list Lv.

So this is a generalization of k coloring. k coloring is the case where Lv equals 1 through k for all vertices. This is, of course, a harder problem. And turns out, it's quite hard. For example, it's NP hard for planar graphs. That's not surprising, because 3-coloring is NP hard for planar graphs.

And size of Lv is less than or equal to 3 for all v. The hardness of planar 3-coloring gives you that. So there's also no natural parameter here, because you can't parametrize by number of colors you have, because even when it's three, this problem is hard. So we're going to parametrize by something else, namely a quantity called outer planarity.

If you know what tree width is, you can think tree width. But tree width is a bit messy to define, so I'll stick to outer planarity, which for planar graphs is within a constant factor. So outer planarity-- if you have a planar graph, this would be an example of a graph of outer planarity 2. Let me draw you an example of a graph of outer planarity 1.

Suppose all of the vertices are on the outside face of your planar graph, or all the vertices are on one face. Then that's an outer planar graph, or 1 outer planar graph. If you have a graph where there are vertices on the outside face, and if you remove all of those vertices from the outside face, the remaining vertices are all on the outside face, this is 2 outer planar.

In general, if you have to remove the vertices on the outside face k times before you're left with no vertices, then your graph is k outer planar. And that k is your outer planarity. So this is an example of a problem that's-- there's no natural parametrization because it's not an optimization problem.

So we're going to throw in a parametrization that often works out quite well. Usually if you take planar graphs parametrized by outer planarity, they are fixed parameter retractable. For example, k coloring, parametrized by outer planarity, is FPT. But list coloring is not. Question?

AUDIENCE: Doesn't the outer planarity depend on the embedding?

PROFESSOR: It depends only slightly on the embedding. I think only by an additive of 1 or something. So it won't matter from a parametrization perspective. Definitely within a constant factor. Good question.

So what we're going to show is that this problem parametrized by outer planarity-- so one note is it is in XP. There is an n to the outer planarity and to the k algorithm using the bounded tree width algorithms, which I won't go into. But we will show that is w1 hard, and assuming ETH, there's no f of k n to the little of k algorithm.

So here's an example of a planar graph problem where we do not get square root of k, which I think would also answer your earlier question. And this is the reduction. It's a reduction from grid tiling. So the idea is if you have a k by k grid, we're going to make something like a k by k grid graph.

Now we have to represent the choice here. We're given a set of pairs for each of these grid cells, which we're now representing as a vertex. But conveniently, we have a choice aspect here. So this is not-- this a thin [? veal ?] for grid tiling. We have this list, Lv, for each vertex of valid choices you could make for that vertex.

So we're going to let L uij-- so this is reduction from grid tiling. L sub uij equals sij. So that's the choice that happens. For every vertex, we have to choose one color.

That mimics the fact that every grid cell in the matrix, we have to choose one item from sij.

So most of our work is already done. Now we have to constrain. If you look at two adjacent vertices, if they're adjacent in a row, then they must have the same first coordinate-- I can never remember. They should have the same second coordinate, sorry, if they're in the same row.

And if they're in the same column, they should have the same first coordinate. So these vertices, which are called v for the vertical connections and h for the horizontal connections, are going to achieve that. And it's a little bit tedious to write down. Basically, let's say between uij and ui plus 1j, so those two vertically adjacent vertices, we're going to add a vertex vijcd for two colors c and d, with list of size 2 c,d for all colors not agreeing on first coordinate.

Again, this is hard to draw in the figure, but easier to write down. So there's a lot of these vertices in between two adjacent ui vertices. There's going to be a bunch. They're parametrized by two colors,cd. Remember, colors are pairs of things, but colors correspond to the items that are in the sijs.

But don't worry about that so much here, except there are colors that are compatible, that they have the same first coordinate. And there are colors that are incompatible. For any two incompatible colors, which is most of them, the ones that don't agree on the first coordinate, we are going to add one of these vertices whose list is exactly cd, the two incompatible colors.

What that means is, suppose this vertex chooses c. Well, then there's a vertex here with list cd. It cannot choose c, because then that edge would be monochromatic. So it must choose d, which means this vertex cannot choose d. So overall, what this means is that these two vertices must choose a compatible color, because we rule out all the incompatible pairs.

So list coloring you can do a lot, but in particular we can simulate grid coloring-- sorry, grid tiling. We're not exploiting a ton of the structure of grid tiling, but we get a nice result here and it's tight. I didn't write down what you do for the horizontal, but it's symmetric. So that's nice.

This is one of the few planar problems where you don't get a square root. The next two, you will get a square root. But before I get to the actual problem, here's a variation of grid tiling, which is a little tricky to prove hard, but is just as hard as grid tiling. Here we need that in every row-- here's the input, here's the output.

Easier to look at the output. In any row-- sorry, that's a column. In any column, the first coordinate is monotonically increasing. Doesn't have to strictly increase, but you have a less than or equal to constraint. This is less than or equal to this. This is less than or equal to this. 4,4,4. Here they happen to be equal. 2,3,3.

Here they increase a little bit, similarly in every row. 2,3,5-- they're monotonically increasing. 1,2,2, 2,2,3. So this is a valid solution to grid tiling with less than or equal to. That's how it's called.

I will not prove that this is w1 hard. I do have a figure for it. It turns out to be a linear expansion. If you have a k by k grid, we're going to end up with a 4k by 4k grid. That part is important. So we get the same kind of hardness result, no f of k and a little o of k, because we only expand k by a factor of 4.

And this is a description of what the sets are. It's kind of messy, but it effectively forces that the choice on the left is actually equal to the choice on the right, even though it only has the ability to specify less than or equal to. But you-- equal is something like modulo capital n, where n is some really large number, like 10 times little n-- something like that.

The details are kind of messy, so I will instead-- I mean, we could just take this as given. This is, like, a core problem to start with. And you can use it to represent lots of nice planar graph problems and 2D problems. So I have one example of each. These are all from this upcoming book on fixed parameter tractability.

So you saw it here first. This book will come out early next year. So here's a problem. It's called scattered set. This is a generalization, in some sense, of independent set. So let me define it over here.

I would also naturally call it d independent set. It might even be called that in the literature. So in this problem, you're given a graph. And you're given two numbers, two natural numbers, k and d. And you want to find k vertices with pairwise distances greater than or equal to k. Sorry, greater than or equal to d.

So if d equals 2, this is independent set. Independent set says the distance between any pair of chosen vertices should be at least 2. There's no adjacent ones of distance 1. d equals 1 is not interesting. d equals 2 is when this problem becomes hard.

Now interestingly, there are FPT algorithms with respect to k and d. So if k and d are small, this problem is easy. And for planar graphs, there's a subexponential FPT algorithm as well. But when d is unbounded, if it's not a parameter, only k is, then this problem is hard even for planar graphs.

So planar is w1 hard with respect to k, only when d can be part of the input arbitrarily large function of n. And also ETH implies there's no planar algorithm with time 2-- sorry, f of k n to the little o of square root of k.

And in fact, for scattered set, there is an n to the big O of square root of k algorithm. I don't think it's been published, but it's mentioned in the book. And there is-- this is tight. This says there is no n to the little o of square root of k algorithm, even when you get an arbitrary computable function f of k in front.

So this is nice. We're planar, so we lose the square aspect, but it's actually a pretty easy reduction from grid tiling with less than or equal to. So you can see the grid here, k equals 3 in this example.

And here, n equals 5, just like our earlier example in fact. All the information is here, as represented by these little red sticks. So we have a 3 by 3 matrix. In each matrix cell, we have a set of items. In this case, there are two items. In this case there are three items. The items are encoded by where the red sticks are in this subgrid.

This is an n by n subgrid within a k by k matrix. So for every present pair in this set sij, we just add a stick. Now, the stick is a really long path. It's 100 times n. So we have this n by n grid, and then we have these hundred n paths. Now this is still planar. You can put that path in there.

And also, this red path is 100, and these are length 1. Black edges are length 1. So what's shown here is actually satisfying assignment where I choose one of these vertices. In scattered set, our goal is to choose vertices that are very far away from each other. How far?

How far is 301 times n plus 1. Roughly, three red sticks plus one traversal of a subgrid plus 1. Roughly three red sticks. So if I choose this vertex and I want to choose one in here, it's got to be at least three red sticks away.

So I'm going to get one red stick here, one red stick here, and one red stick there. So that's good. But that just says I choose exactly one out of each of these things. Once I choose one of these endpoints, I certainly can't choose another one because it's only two red sticks away. I can only choose one per subgrid.

But then also, I want the lesser and equal to constraint. And that's the plus n over here. So I have three red sticks plus n. That's the 1. Because I have plus n, n is the width, let's say, these guys. So once I choose this guy, I have to be three red sticks- - 1, 2, 3 red sticks away.

But I also need to be an additional n away. And here, I am that because I have 1, 2, 3, 4, 5, 6 away. I'm actually one more than n away. And there's a plus 1 over there, so that's good. I'm 6 away. I need to be 6 away. And that corresponds to this being in the fourth column and this being in the fourth column.

In other words, it corresponds to the second coordinate of this guy being less than or equal to the second coordinate of this guy. So it's exactly the grid tiling with less than or equal to constraint horizontally and symmetrically, vertically. Because the distance between a point here to point here is going to be go straight down, jump, use the red stick, and then go teleport left right and then go straight down from there.

So that distance corresponds to exactly when all the less than or equal to constraints are satisfied. If and only if, so this a reduction from grid tiling with less than or equal to to scattered set. [INAUDIBLE] is w1 hard. Now here, notice k prime is k squared, because we are choosing one vertex per matrix cell.

And they're k squared cells. So here we are losing the quadratic blowup. Questions? One more similar example. There aren't a ton of hardness results here, so not a lot to choose from. There are some multi-way [? Kant and ?] other things. But among simple examples, here's another simple example-- very similar looking.

This is a graph in 2D plane, so to speak. You can also think of it as a unit disk graph problem. So a unit disk graph is I take some points in the plane, 2D coordinates, let's say given by rational values. x and y-coordinates are given by rationals.

And if I look at any two vertices, if they live-- if the distance between them is less than or equal to 1, then I add an edge. So here, I might have a graph-- this is going to be a lot of edges with that notion of 1.

You can have big cliques in a unit disk graph, but it's kind of planar-ish. Especially if you have distant vertices, they're not going to be connected. So you have these local cliques, but they're kind of connected in a planar-like way. Definition clear? Edge if and only if distance at most 1.

So what about independent set in unit disk graphs? We know independent set in general's hard. Independent set in unit disk graphs is almost as hard. Again, there's a quadratic loss in the parameter. But problem is, w1 hard and has the same kind of hardness as scattered set.

By similar kind of structure, here I'm actually giving you the grid tiling with less than or equal to. Probably also corresponds to the last example, but here it is in the independent set unit disk problem. Independent set in a unit disk is the same thing as choosing some vertices, like this one, as the center of a radius one half disk and then those radius one half disks should not intersect each other. Because these two things will have distance at least 1 if and only if radius one half disk and a radius one half disk here do not intersect.

So it's really about choosing centers for these disks that don't hit. And so again, what we're going to do is imagine an n by n subgrid within each cell of the matrix. But not all of those points are actually in the set, only the ones that are in-- in this case, s1,1 these pairs of guys as written here, so 1, 1 is in there. I guess that's that guy. 2,5 is this guy. 3,3 is that guy. Those points we'll actually put in the problem.

These tiny dots are just place markers. There's no actual point there. Then we construct the unit disk graph on the structure. And again, if we set the unit right, and these are super tiny, in the same way that we had the red edges, which were, like, 100 times longer than the small things over here, we're going to have, let's say this thing is 100 times smaller than that distance, enough so that these circles act kind of like squares.

If you look very close to here, this looks straight. So these are very compressed, so it's probably going to be more like a factor of n smaller. These effectively act like squares. It's just a matter of whether the horizontal extent of this disk hits the horizontal extent of this disk. And that is a less than or equal to constraint.

Once you choose something in column 2 here, the next one has to be column at least 2. Here, there there's a gap because we chose column 3. Here, there's a gap because we chose column 5. But for example here, we chose column 2, column 2 and these guys are almost touching. But barely not touching.

And as long as you have the less than or equal to constraint on the columns, then you'll be OK. The disk won't intersect and it's an if and only if. So again, we represent grid tiling with less than or equal to and independent set or disk packing problem in the plane. It's kind of cool. Questions?

All right. Well-- oh, more fun facts. So we have for independent set in unit disk graphs, we have that there is no f of k n to the little o of square root of k algorithm. There actually is an n to the big O of square root of k algorithm.

We also get-- and I won't go through this. I think it's pretty trivial, based on what I said last class. But we get that there's no efficient p test for this problem unless-- sorry. This we definitely get from last lecture. I said if you're w1 hard or FPT-- if you're w1 hard and FPT does not equal w1, then you are not FPT.

If you're not FPT, there's no efficient p test, no f of 1 over epsilon times n to some constant. In fact, if you assume ETH, you get an even stronger form of that. And in this example, we get there is no 2 to the 1 over epsilon to the 1 minus delta power times n to the order 1 1 plus epsilon approximation. This is the original result from the Daniel Marx paper that introduced-- this was his motivation for introducing grid tiling.

So again, you get-- out of all these lower bounds, you also get results about inapproximability, which is another reason to care. Even if you don't care about parametrized complexity, these are the only ways known to prove lower bounds on how slow your p test has to be. Because there are p tests for these problems, but only so efficient. That's it for fixed parameter tractability. Next class, we'll do something completely different.