PROFESSOR: Today we have a lecturer, guest lecture two of two, Costis Daskalakis.

COSTIS DASKALAKIS: Glad to be back. So let's continue on the path we followed last time. Let me remind you what we did last time, first of all.

So I talked about interesting theorems in topology-- Nash, Sperner, and Brouwer. And I defined the corresponding-- so these were theorems in topology. Define the corresponding problems. And because of these existence theorems, the corresponding search problems were total.

And then I looked into the problems in NP that are total, and I tried to identify what in these problems make them total and tried to identify combinatorial argument that guarantees the existence of solutions in these problems.

Motivated by the argument, which turned out to be a parity argument on directed graphs, I defined the class PPAD, and I introduced the problem of ArithmCircuitSAT, which is PPAD complete, and from which I promised to show a bunch of PPAD hardness deductions this time.

So let me remind you the salient points from this list before I keep going. So first of all, the PPAD class has a combinatorial flavor. In the definition of the class, what I'm doing is I'm defining a graph on all possible n-bit strings, so an exponentially large set by providing two circuits, P and N.

P is a circuit of possible father, and N is the circuit of possible child. And given these two circuits, I establish a directed edge between string v1 and string v2, if they agree on their parent relationship, meaning v2 believes V1 is his father, and also v1 believes v2 is its child.

In that case if this condition is true, I establish an edge between these two pairs of nodes. And I do the same for all pairs of strings. And in the end, I get a graph. And the problem end of the line is given these two circuits and the corresponding graph that they define on this set, if the all 0 string is unbalanced meaning different in and out degree, then I want you to find another unbalanced node, string, which is guaranteed to exist by the parity arguments.

And PPAD is a class of all search problems in FNP they are reducable-- polynomial time reducible to this problem. I'll remind you also of structure of the graph defined by these two circuits. It's easy to verify that if my edge definition is this, then every vertex has in degree and out degree at most once.

So the graph that's induced by these two circuits will have this form. And basically, I'm looking for all red points. For any of these points, any of these red points, strings, are solutions, any unbalanced string, except for the all 0 string.

If the all 0 string is not unbalanced, then I don't want you to do anything. If it is unbalanced, then I'm looking for any of these red vertices. So that's the class PPAD, and it has a combinatorial flavor.

So I'm defining a huge graph, we have these two circuits. And I'm asking you to find any of these red points. So on the other hand, sort of like the problems we were targeting, Nash equilibrium, Brouwer's theorem, had a more continues flavor.

So instead of working directly with this problem, with the end of the line, and trying to reduce this problem too Nash and to Brouwer to establish PPAD hardness deductions, I actually introduced a problem that actually closer to this problem. It has a continuous flavor.

And it was the problem ArithmCircuitSAT. There were a bunch of problems last time about the definition of the problem. So I decided to be more explicit about what it is.

So basically I'm giving you a circuit that has two types of nodes, viable nodes, v1 through vn, and gate nodes, g1 through gn. Now gate node has one of six possible flavors. It could be an assignment gate, and addition gate, a subtraction gate, set

equal to constant gate, multiply by a constant gate, and comparison gate.

Depending on the type, it's going to have from 0 to the 3 inputs, to 2 inputs. And it always has 1 output. Now what I wanted to emphasize is that this graph doesn't have inputs-- input variables. Loops are allowed. And what I want to emphasize that I didn't emphasise last time is that variable nodes have n degree 1, and gates have 0, 1, or 2 inputs, depending on their type.

Otherwise for instance, the out degree of a node could be arbitrary. The fan out could be arbitrary, it doesn't matter. But you have to respect this. Every variable node has n degree 1. Every gate node has 1, 2, or 2 inputs depending on the type.

There are no edges between gates and gates, and variables and variables. There are only edges between variables and gates, and gates to variables. That's the input. The input is a circuit that has this form.

And what I want you to do is I want you to find an assignment of real 0, 1 values to the variables of this circuit such that the constraints of the gates are satisfied. And here are the constrains of the gates.

So if they gate is an assignment gate, I want the output node, the node who's connected to the output of the gate has equal value to the node that's feeding into that gate. If they gate is an addition gate, I want that variable node who's connected to the output of the gate to be basically the sum of the values of the inputs to the addition gate, except I'm also going to threshold it. I'm not going to allow it going above 1, and so and so forth. So these are the gate conditions.

So now what I said last time is that a satisfying assignment always exists for this problem. It's not a priori, so it a requires work and actually it is going through a fixed point to argue that there is always a solution to this problem. What I also claimed last time is that it's PPAD complete to find a satisfying assignment. So it's a natural starting point for deductions.

What I also said last time is that in fact, I can allow some noise in the error, in the

gate constraints. I can allow plus minus epsilon deviation from the gate constraints. And so this should be a minus epsilon. Sorry about that.

And this epsilon is part of the input. I can give as input both a circuit and an epsilon. And I will ask you to satisfy the gate constraints with an epsilon. Now last time I also showed you the structure the hardness proof for Nash, which basically took generic PPAD end of the line problem.

Today PPAD complete problem, end of the line, is embedded into geometry, into the 3-D cube. Then we define a version of Sperner's Lemma, which then introduced ArithmCircuitSAT. And I didn't show this part of the deduction, and I'm not going to show it because that's the complicated part.

But I am going to show is how to go from this problem, the ArithmCircuitSAT to Nash equilibrium, just to show how easy it is to work with this problem and reduce to other problems. So that's where I want to focus on. So that's the review from last time. And this time, I want to talk about-- I want to show the PPAD completeness of Nash equilibrium.

I'm going to briefly give two other examples that have come from combinatorics. And then lastly, I'm going to talk about other existence arguments and the complexity classes that they define. So I'm going to introduce these classes, PPA, PPP, and PLS.

Before I do that, I also thought it was a question from last time. PPAD stands for polynomial parity argument in directed graphs, corresponding to the fact that this class is defined with this parity argument in directed graphs in mind.

So let's focus on this reduction, from Circuit to Nash. I want to introduce a concept before I show the reduction. That concept is graphical games and polymatrix games, a special case of graphical games. So graphical games were introduced in 2001 by Kearns, Littman, and Singh as something very natural.

Basically they tried to capture situations where the payoff of a player only depends on the actions of a few other players, because of geographical, communication, or

other constraints. So in the graphical game, the players are nodes in the graph, in a directive graph. And a player's payoff only depends on her own strategy, as well as the strategy of the players that point to him.

For example, this guy's payoff depends on this guy's, this guy's, and this guy's action. Because all of these guys point to him, as well as his own action. A special case of these games was actually introduced much earlier.

So polymatrix games are graphical games where the payoff functions of the nodes are actually edge-wise separable. So for instance, the payoff of this guy as a function of everybody's mixed strategy is separable overall edges that point to him of some pairwise player function that has to do with his action and his neighbor's action. That's what's written here.

And it's not very hard to see that any-- so this is a utility function that depends on the two mixed strategies. And by assumption, these players randomize independently of each other. So any such expectation of a pair of players' strategies that's are a product can actually be written as a quadratic form.

Do you see that? Let me write on the board. So again, Xv is the mixed strategy of player v. Xw is the mixed strategy of w. What I mean by Uwv Xu, comma Xw is basically an expectation over an action Su drawn from Xu, and action is Sw.

Sorry-- Sv drawn from Xv, Sw drawn from Xw, independently of-- and that's just the sum of all the Us and all the Sws of U, W, sv, sw, and then the probabilities. And that's a quadratic form. So that's what I mean by this line.

Because players play independently from each other, any expectation with respect to that product distribution is a quadratic form. I'm not saying something interesting. Good? So then a polymatrix game is really, because of this, defined by a directive graph.

And then for every directed edge, and every-- there is a matrix that defines the quadratic form for that edge. So polymatrix game is easily described. You can describe by specifying the graph and then giving a matrix what every directed edge.

So bimatrix game are two player games. So that's why these are called polymatrix, because you have many matrices. In a two player game, you only need to give two matrices, one for a two player game is just player 1, player 2. And you give one matrix for this direction, and one matrix for that direction. So that's a bimatrix game. This is a polymatrix game.

Now what I want to do is I want to-- in order to reduce ArithmCircuitSAT to Nash, instead what I'm going to do first is I'm going to reduce ArithmCircuitSAT to finding a Nash equilibrium in a polymatrix game. That's what I want to do first.

After I do that, then I'm going to reduce it to-- so here I have many players. I want to go down to two players. But the first step is to just go to multiplayer Nash equilibrium. Then that deduction is the easy part. Also this is an easy part.

The hard part happened before. So now how can we reduce an ArithmCircuitSAT problem into a polymatrix game Nash equilibrium problem? Like in LP completeness, like in reductions for NP, RP hardness proofs, you have to give gadgets.

You have to identify objects in polymatrix games that simulate the operations that happen in your circuit over here. So what I want to introduce is what is called a game gadget. These are small polymatrix games that do various arithmetic operations, which I can then put together to simulate an ArithmCircuitSAT problem.

So what I want to do is I'm going to give you a flavor of these gadgets. So I'm going to give you the addition gadgets. I want to give you a polymatrix game that does addition. So it's going to have-- this game is going to have four players. Everything player will have just two strategies-- 0 and 1, two pure strategies, hence the mixed strategy of that player is going to be a real number in 0, 1, which corresponds to the probability by which this player plays 1.

So here's the structure of the gadget. So this gadget, I'm showing it here embedded into a potentially bigger polymatrix game. But the gadget itself is going to have four

players, X, Y, W, Z. Now X, Y are what is called the input to the gadget. So these are players who point to W, but don't depend on the actions of W and X because there are no directions going the other way.

So these guys don't care about what these players are doing, and they serve as an input to the gadget. Now this player, W, gets his input of players strategies in this gadget. While X only cares about the W is doing. So X and Y are going to be called the input players to the gadget.

Z is going to be called the output player to the gadget. And W is going to be called the auxiliary player. Now what I want to do is I want to define-- the only thing I'm going to define is the payoff function of W and the payoff function of Z.

And I'm going to define this payoff functions in a way that addition somehow happens at a Nash equilibrium of this little game. So I'm going to define the payoff succinctly. But then I'm going to convince you that these really correspond to tables. So I'm going to say that W is paid an expected-- depending on what he plays.

So if he plays 0, he gets paid an expected probability X equal $1, plus probably X equals Y equals $1 if he plays 0. But if he plays 1, he only gets paid probabilities equals $1. So in some sense if he plays 0, he looks to the left. If he plays 1, he looks to the right.

His payoffs are the sum of these two guys probabilities of playing 1 in one case, and this guy's probability of playing 1 in the other case. So that's the payoff function to W. Now you would ask me maybe how can you implement this payoff functions.

And that's actually very easy. So here's the table for player W. So when W plays 0, his payoff is depending on the strategies of X and Y. I'm going to define it to be 0, 1, 1, and 2.

Now notice that in expectation over X's and Y's strategies, he's payoff-- so if he plays 0, his expected payoff over X and Y's strategies is just the probability that X plays 1 plus the probability that Y plays 1. Do you see this from this matrix?

So what's the expected payoff to W when he plays 0? So this is 1 if Y plays 1, but X plays 0. And it's 1 if X plays 1, but Y plays 0, and 2 if they both play 1. And I claim that if you properly collect the terms, this is just equal to probability Y plays 1, and X plays 1. So that's what I claim.

So I covered this line. This line is also easy to cover by saying that when W plays 0, his payoff just depends on what Z is doing. And it's going to be like this. So when W plays 1, his expected payoff is exactly the probability Z plays 1.

So what I've written here is actually consistent with some tables that I'm hiding from this slide. So that's, I guess, that's what I want to write. And then similarly, Z is paid to play the opposite of W. What do I mean by that?

I mean that Z's payoff, when he plays 0, is exactly 1/2, no matter what W does. But if he plays 1, his payoff is 1 minus W plays 1. Which again, you should be able to see that there's a table, a little table implementing these payoff functions.

And that sounds weird. Why did I define it this way? Here's my claim. In any Nash equilibrium of a game that contains this little gadget, the probability that the output player plays 1 is basically the sum of the probability that the input players play 1 thresholded at 1 of course.

So if this little gadget that I define here is part of a bigger game-- now what can the bigger game do? So it can fit something into X. It can take the value of Z and use it in some other way, potentially looping around, doing anything and once, except the only inputs to W's payoff are X, Y, and Z.

And the only input to Z's payoff is W, but otherwise the game can be arbitrary. So if this game that I define here is embedded within a bigger game, then in anything Nash equilibrium of that bigger game, the probability that this guy plays 1 is exactly the sum of the probabilities that these two guys play 1, thresholded at 1.

Now how can we see this? Why is that true? It's a little case analysis. It's very simple. Let's try to do it. So suppose that the probability that Z plays 1 is smaller than the probability X plays 1 plus the probability Y plays 1.

Actually, let's do something else-- it's smaller than the min between this and 1. What happens in this case? So what happens if Z is smaller than the minimum of these two values? What is W going to do in that case?

AUDIENCE: Is he going to play 0?

COSTIS DASKALAKIS: Yeah, because 0 gives him a high payoff, gives his a higher payoff than this guy, but because of that condition. So this implies that W is going to play 0 with probability 1. Now what does this imply?

If W plays 0 with probability 1, what does Z do? If he plays 0, he gets 0.5. What if he plays 1? How much does he get? He gets 1 because of that condition. So he's going to play what? 1. And so this implies that-- but how can 1 be smaller than the minimal of 1 and something else?

That's can't be. So this is impossible. So do the other side, or maybe you already trust me that it's OK. Let's try to argue that this cannot be the case. I don't need to do the mean anymore because it can't possibly be that this guy plays more than 1 probability.

So this is the only case I want to consider now. So what happens in this case? The same logic. What does W do?

AUDIENCE: Play 1?

COSTIS DASKALAKIS: Has to play 1 because that's the better strategy. And what does he do?

AUDIENCE: Play 0.

COSTIS DASKALAKIS: But 0 can't be bigger than the sum of two r probabilities, so this and that can happen. Hence, the only case that's possible is this one. So it's very simple. So the only realization that one had to do is that-- where the arithmetic happens?

It happens in the mixed strategies chosen by the players. That's where it happens.

So this is the addition gadget. Similarly, one can define other gadgets. For example, the subtraction gadget is really, really similar to this except with one change.

I replace this plus with a minus. That's the only change I need to do. I'm not sure I want to do it, but in other words, this table-- I only change this table to have 0 here and minus 1 here. And I've exactly implemented that, the subtraction. And the proof is exactly the same.

You can argue that in any Nash equilibrium of a game that contains this gadget, the probability that this guy plays 1 is actually the difference of the probabilities that X plays 1 and Y plays 1, a probability truncated at 0. Any questions about what taking place here? What we're trying to do is we want take a ArithmCircuitSAT instance, and we want to create a polymatrix game that simulates that circuit SAT instance. And what we need for that is gadgets, so little polymatrix games that implement various operations.

And I showed you addition and subtraction. But I claim that you can implement a bunch of other gates. To have a more succinct notation, I'm going to use probability that some node plays 1. I'm not going to differentiate that with an X, and the probability that X plays 1.

With this notation, I claim that we can implement all the gadgets that we need to simulate a circuit SAT instance. So let's look at this table. So in all these lines of this table, Z is the output player of the gadget, X and Y are the input players of the gadgets. And potentially, the gadget's implementing each of these gates use auxiliary players.

Like in the addition case, I was using one intermediate player. He was the auxiliary player. And the claim is that you can implement gadgets such that if any of these gadgets is contained in a bigger polymatrix game, then any Nash equilibrium of these bigger polymatrix game these conditions are satisfied by the output and input players of the gadget.

And this is as long as you don't mess up with the gadgets, meaning that the bigger

the game can have edges into the input players of the gadgets, and edges out of output players of the gadgets. But I claim that you can implement all of the gadgets that you need.

So in particular, if you have an instance for ArithmCircuitSAT you can create the polymatrix game by composing gadgets for each of these gates, so that at any Nash equilibrium of polymatrix game, all gates conditions are satisfied. So in particular, you are-- by finding a Nash equilibrium of this game, you are solving the instance of ArithmCircuitSAT you started with.

So that's the idea. Any questions about what happened so far in the lecture? Questions about this deduction, or how the gadgets work? Yeah?

AUDIENCE: Does this give a valid proof that every instance of ArithmCircuitSAT has a satisfying ascendant?

COSTIS DASKALAKIS: Yeah, that's correct. Because this is a game. Some by Nash's theorem, there is a Nash equilibrium. Hence, there is a solution to that. Because a Nash equilibrium is a solution to that. That's exactly right. Actually that's how we get it.

One thing that's not exactly obvious, but you can argue that it's true is that actually not only there's always a solution, but there is a solution in irrational numbers with polynomial description complexity in the input size. So that requires some linear programming techniques. But that's also use it to show.

A priori, Nash's theorem will give you that any solution exists. It wouldn't give guarantees about the description complexity of that solution. But using linear programming, you can argue that there's always a rational solution with polynomial description complexity.

Are you interested in seeing any of these gadgets? I've only shown addition. Do you want to see anything else? Do you believe me that I can actually do that?

AUDIENCE: The comparison one that has arbitrary output for a certain conditions seems interesting.

COSTIS DASKALAKIS: OK, I can show that. So let me try to do the comparison gadgets. They're actually simpler than the addition gadget. So I want to implement a comparison. So I'm only going to have the input players and the output player.

And the payoffs of the output player are-- let's see. So if you play 0, then his payoff is only depends on the X. And it looks like is. If he plays 1, let's see if it works. I haven't prepared this but it's easy to figure this out. Worse case, I'll back track.

So now what I want to argue is that-- let's bring up the conditions that I need to satisfy. So I want to satisfy those conditions. So let's see if I do satisfy these conditions. So I messed it up a little bit, I guess. So I have to replace X here and Y here.

Now let's see if that's true. So I claim that if probability x is bigger than probability y, then what do I prefer to play? I'm prefer to play 0. Similarly for the other case, when they're equal, anything is possible. It's that's simple. There's not [INAUDIBLE] going on here.

Cool. So I've established that this direction, this part of the direction. Now I want to go down from a polymatrix game to a two-player game. That's the next part of the lecture. How do you go down to two players?

Well the first thing to note is that all the gates can be implemented with bipartite graphs. So if you remember my addition gadget, it had the input and the output players on one side and the auxiliary player on one side. My comparison gadget does not satisfy this probability because the input players and the output players are on different sides.

But I claim that you can implement actually a comparison by adding an X and an additional step with a bipartite graph. So I claim that all gates can be implemented with polymatrix games that have input and output players on one side, and auxiliary vertices on the other side.

So I claim that all my gadgets are actually bipartite. So in particularly, I can color this

graph with two colors, blue and red. And what I want to do is I want to create these super players. I'm going to call the lawyers.

I'm going to have a red lawyer and a blue lawyer. And the red player is going to represent all the nodes that are colored red in here, while the blue lawyer is going to represent all blue nodes.

And what I want to do is I want to define the lawyer game in the next slide. So every lawyer, so the red lawyer, his strategy set is the union, not the product, the union. That's important, otherwise my deduction would be exponentially large. My induction wouldn't be a polynomial.

So as a lawyer, and that poses actually technicalities in the construction that I'm about to present. But every lawyer will have a strategy set, a pure strategy set, the union of the pure strategy sets of the clients that he represents. So the red lawyer has one strategy for every strategy of every red nodes.

The blue lawyer has one strategy for every strategy of every blue node. Now what I want to do is I want to define the payoffs in the lower game. What are the payoffs in the lower game?

Well, what happens if the red lawyer decides to play strategy i or red client U? And what if the blue lawyer plays strategy j of blue node v? What are the payoffs of the two lawyers in that choice of strategies?

Well, it's going to be exactly the corresponding payoffs of the nodes of the polymatrix game. So the red lawyer will the payoff that U would have gotten in the polymatrix game if these two players played i and j. And the blue lawyer's going to get his client's value, so v's value, under the same choice of strategies.

So in particularly, if there's no edge between those two nodes that blue and red lawyers chose to play, then nobody gets anything. The payoff here is going to be 0. And if there is a directed edge from V to U, then this is going to be 0, and that's whatever it is, and vice versa.

But you get the idea. If the blue lawyer decides to represent a client, and the red lawyer decides to represent another client, then the payoff of the lawyers are the corresponding payoffs that the clients would have gotten had they played the strategies that the lawyers decided to play for them. That's the definition of the lawyer game. Is this clear?

Now the wishful thinking is that if X and Y is a Nash equilibrium of the lawyer game, then if I look at the marginal probability distributions on the different nodes, on the different clients by their lawyer, then this marginal probability distribution are a Nash equilibrium.

So the wishful thinking is that if I start with a Nash equilibrium of the lawyer game, then if I look at the distribution that the red lawyer, the marginal probability distribution that the red lawyer places on every red node separately, and the marginal probability distributions that the blue lawyer places on all blue node separately, so all this collection of marginal probability distributions, that this collection is a Nash equilibrium of the polymatrix game. That's the wishful thinking.

Questions about what the wishful thinking is? Of course, this is wishful thinking because we know how lawyers behave. They only represent the lucrative clients. There's no reason that the red lawyer would have incentive to choose at least some strategy to place positive probability mass on every node he represents.

Maybe in the Nash equilibrium of this lawyer game, some of these marginal distributions are actually undefined because the lawyers-- are ill defined because the lawyers place 0 probability mass on the strategies of those nodes. So that wishful thinking isn't going through.

But there is some truth to that. So there is a way to fix it. So here's how we're going to fix the lawyer game. So we know what lawyers like. They like money. So we're going to define a high stakes game.

But the lawyers are going to play on the side at the same time as the actual game that we're interested in. So for a lot of cash-- so those are the stakes of that game.

We're going to do the following. So some terminology first.

So suppose that we had lots of generalities. Suppose that every lawyer has n clients he represents. And let's label the red lawyer's clients 1 through n, and the blue lawyer's clients 1 through n. If you know one of the two lawyers have fewer clients, then we can pad this with dummy players and that doesn't change the polymatrix game's equilibria.

So suppose that both lawyers represent the same number of players, clients. And let's label both lawyers' clients one through n, in an arbitrary way. The strategies of the high stakes game are exactly the same as the strategies of the game that I showed in the previous slide.

In particular, the red lawyer has the union of the strategies of blue nodes. And red lawyer has the union of the strategies of the red nodes. Now what's the high stakes game? Suppose that the red player plays any strategy of client j, and the blue lawyer play any strategy of client k.

Then if they choose different clients, they both get 0 dollars. But if they choose the same client-- I mean, it's not the same client, it's the same label of a client because they each represent different clients.

But if they choose different labels, then they both get 0. If the choose the same label, whoever it was, the red lawyer gets a lot of cash, and that blue lawyer loses a lot of cash. Again, so this game has the same strategies as the lawyer game that I showed in the previous slide.

Except now in this game, all the lawyers care about is the labels of the clients whose strategies they choose. So if they choose a strategy of a client that has the same label, then the red lawyer gains a lot of money, and the blue lawyer lose a lot of money.

But if they choose strategies of clients with different labels, then they both get 0. In other words in some sense, the blue lawyer is trying to avoid the red lawyer, and the red lawyer is trying to catch the blue lawyer. So now this is a simple 0 sum game.

And it's not hard to see that in any Nash equilibrium of this game, the lawyers are going to represent every client with the same probability. So each lawyer assigns a probability exactly 1/n to the set of his strategies corresponding to each of his clients.

So the high stakes game has the property that the lawyers represent all their clients with the same probability distribution, and you can divide the probability distribution in an arbitrary way within the strategies of each of their clients in this game. That's easy to see just by the symmetry of this games, that has to be true.

And with these two definitions, the game that I'm going to do for my reduction from polymatrix games to two player games is going to be the sum of these two games. So this is the game that I defined earlier. And this matrix is a block matrix.

So this is a block of strategies corresponding to-- this matrix is a matrix that is a constant matrix where everything is big. And this block corresponds to-- these are the strategies of client with label 1 for the red lawyer. And this is the strategies of the client of the blue lawyer with label 1, and so on so forth.

So this is a block matrix. And there are ms and minus ms in this diagonal blocks, and everything else is 0. So that's the high stakes game. It's played along blocks of strategies because all I care of is the label the client I'm choosing for each of these lawyers. Well, this is a more fine-grained game, where I not only care about which clients and choosing, but also which strategies of these clients I'm choosing.

And I'm going to choose an m that overwhelms the payoffs in this games. So this condition is OK for what I'm about to say. But think of m as huge compared to the maximum utility in this game times the number of clients in that game.

AUDIENCE: Just to make sure I'm understanding this so far-- so in the naive game, if the red guy chooses strategy and the blue guys chooses a strategy and the two vertices they choose from are not connected by an edge, then they both get 0.

COSTIS Yeah. And if they're connected, they get payoffs that the corresponding nodes

DASKALAKIS: would have gotten--

AUDIENCE: [INAUDIBLE] vertical edge, one is the one on the outgoing vertex will still get 0.

COSTIS DASKALAKIS: Exactly, yeah. Any other questions about-- yeah?

AUDIENCE: Even with a really large choice of m, can't it still mess up the values a little bit?

COSTIS DASKALAKIS: Yeah. It will mess it up a little bit. So the Nash equilibrium of this game, the addition of these two games, is not going to have the property that this game by itself had. But it's going to be very close. And I'm going to be specific about it.

AUDIENCE: So the choice of me is so that the isn't that big.

COSTIS DASKALAKIS: Even for arbitrarily large m, I can't actually quite show that the lawyers represent their clients exactly equally. But I can bring it close. And the larger m is the closer the come.

In particular, I can show the following statement that in any Nash equilibrium of the combined game, if X used the total mass that the red lawyer places on the union of strategies of node U, then that's about 1/m. But there is the matter that's the case with m, and similarly for the blue lawyer.

So approximately they're representing all their clients. And if I choose m huge then at least my marginals are well-defined now. So there is probability mass on every client and I can define these marginal distributions. Now whether these distributions are useful or not, we are about to see.

But what I'm saying here is that as far as deciding how to split the pie into my clients, really the large game is what matters. The high stakes game is what matters. Because if I make a mistake, m is huge, some huge [INAUDIBLE]. So for splitting, for deciding how to split my total unit of probability mass into my clients, only the high stakes game-- essentially, only the high stakes games matter.

On the other hand, when it comes to having decided how much mass to put on the

unit of a particular node strategies, deciding about the fine-grained decision of how to allocate that Xu into the different strategies on that node U, then actually only the small game matters. And the reason is that the payoff different of the red lawyer from strategies Ui and Uj-- so to distinguishes between strategies i and j of node U, the payoff difference between these two choices doesn't have m in it.

If you actually look at it, m goes away. There's no m in the payoff difference between these two options. So which one is better doesn't have m in it, essentially doesn't have m in it. There is some m in here because this I'm summing over all nodes, and different nodes have different sum of probabilities.

But I'm going to get to that. But trust me that when the red lawyer is trying to decide how to allocate this Xu probability that he has decided to allocate on the difference strategies of node U, he looks at this difference, that's the difference in the two payoffs.

And essentially there's no m in this equation. So we're going to see when m gets into the picture and why. But this is just from the definition of the game. Again, this is exactly true. The difference of the two payoff is exactly this.

Now from that equation, it follows that if they lawyer decides to put positive probability mass on a particular strategy i of that node, then for all js, for all alternative strategies of that node that he could choose, it better be that this payoff difference is positive. And that really, really looks like the condition of the Nash equilibrium conditions for the client, if you think about it.

Because if I define the marginal probabilities of node U on these two strategies, on these strategies, then this is really the Nash equilibrium condition for that node U. The only problem is to go from this equation for the unmarginalized probabilities Vy, Vl's, to the marginals. I'm dividing with something, and that something is an equal for all these.

So if it was equal, then these marginals would actually be directly Nash equilibrium. Because this condition I could just divide by 1/n, and this would be made marginal

probability. And that's exactly the equilibrium condition for the client U.

The problem is that these Xvs are not all equal. Yvs are not equal. So I cannot just divide by 1/n and claim that the equation is still true. But the point is that this error doesn't create too much problems in the equilibrium conditions. And OK, it's not going to be an exact Nash equilibrium for the polymatrix game, but it's going to be an approximate one.

And because I can take n to be as large as I want, I can make this approximation go to 0 as fast as I want. And remember this ArithmCircuitSAT problem allowed some error in it. So what effectively is going to happen here is that I'm going to get an approximate equilibrium of the polymatrix game.

So that would correspond to an approximate equilibrium of the arithmetic SAT problem I started with. But this approximation can back accommodated, and the problem is to PPAD hard. So that's approximately how the argument works.

The fact that the disagreements aren't uniform doesn't really matter as long as n is chosen large enough. I went a bit too fast. I didn't mean for this to be very detailed. But I meant to convey the bigger picture. And the bigger picture basically says that you can analyze what happens in this lower game in two steps.

In one step, you argue that the lawyers approximately represent all their clients. In the other step, you have to decide how these lawyers allocate their probabilities to the different strategies of a particular node. That leads you to write down this difference of payoffs that the lawyer is experiencing when he switches. This is tracking how much better the expected payoff from this strategies is compared to that strategy.

And by the equilibrium conditions of the lawyer game, you get the inequality I showed in the next slide that if the lawyer decides to place positive probability mass to a strategy i of node U, then it must be that there is no alternative strategy, Uj, that would give them the better payoff, so you get this condition. And that's essentially the equilibrium condition for the polymatrix game, except that you need to normalize

these guy.

And when you try to normalize these guy, you run into the problem that the lawyers don't play uniform strategies for their clients. But they play approximate uniform distribution of their clients. So this inequality gets messed up a little bit. So effectively that means that the players are almost best responding. The marginal distributions on an approximate Nash equilibrium of the polymatrix game.

Also because the polymatrix game came from my ArithmCircuitSAT problem, an approximate evaluation of that problem. But theses are all PPAD hard, as I pointed out earlier. That's the high level idea. And trust me, the details are not hard at all. OK

So you just have to trust me that the approximate uniform distribution of the claim are true and that dividing by approximately 1/n here doesn't mess up this condition too much. So this was the end of the proof basically. That's basically how the proof goes.

So if have this, it's easy to do this, and then it's easy to go here. Any questions? Now the reduction I showed you was not from the original paper. It was established by a follow up paper by Chen and Deng. In that original paper, we actually took this problem in reduced to 4-player Nash.

The proof is identical to the one I showed you. The only reason that we went to four players instead of two players is that our gadgets were four-partite instead of bipartite. So we had four colors and four lawyers. Now why did we have four-partite gadgets?

Well, we were being a bit silly. We had in our arithmetic circuit, we had an extra gate, which plus multiplication, not by a constant, but the multiplication of two numbers. And you can show that this-- but we didn't use it. We were being a bit silly.

We had this gadget here that we weren't using. And it was hard to-- actually it's impossible to make it into a bipartite gadget. So in our recent paper, we made it into a four-partite gadget. And then, in the followup paper we actual managed to reduce

it to a three-partite gadget.

But then this multiplication gadget actually you can show cannot be reduced to bipartite. So these guys observed that actually we are not using multiplication. So you can implement all gadgets that we actually used with partite graphs. So you can go down to two lawyers instead of three lawyers. So it felt like leaving money on the table, but that's OK.

That's my discussion of PPAD-completeness of Nash. Yeah?

AUDIENCE: Obviously, in zero-sum games, it's easy to find an equilibrium. Is there something in between zero-sum games and general two-player games for which any hardness result is known?

COSTIS DASKALAKIS: Yeah. A natural way to interpolate between zero-sum games and general two-player games is the following. So in a zero-sum game let's call R and C the payoff matrix of the two players. In the game of zero-sum, if R plus C, the sum of these two matrices is identically 0, as I claimed in the very first slide I gave last time.

Now you could call a game rank-r if R plus C has rank-r. Now what is known is that rank-1 games can be solved in P. I believe that it's known that rank-3 games are PPAD hard. And you should look at the paper by Mehta, Ruta Mehta in '14, I believe, where she shows that.

She might even be showing rank-2, but I'm not sure. Maybe there is some gap there. Maybe it's even-- so there is a constant where it's already PPAD hard. And while I'm here, an interesting open question is approximate equilibrium. So interesting open problem-- approximate Nash equilibria.

Even into two-player games, we've don't know how to find them. So let me remind you what this. So a Nash equilibrium is a pair of strategies such that no one has an incentive to change his randomization given what the other player is doing.

And epsilon Nash equilibrium is when this condition are true to within an order of epsilon. So no player has incentive, more than epsilon, additive epsilon incentive, to

changed his strategies. So at most, additive epsilon incentive to change.

Now what do we know about these problems? Well, if the input to your problem is a game and an epsilon, then it follows from those results that that's PPAD complete. Now it's even true that if your input is a game, and you have a prespecified epsilon that's inverse polynomial in the size of the game, so that the input is a game, and there is a inverse polynomial function, and that's your epsilon.

And you want to find an inverse polynomial Nash equilibrium of this given game that's still PPAD complete. But what is not known is epsilon constant. How hard is it to find equilibria when epsilon is a constant? And if your matrices have entries in 0,1, then we know how to do this in time n to the order log n over epsilon squared.

We know that there is no FPTAS for the problem because of actually these results, follow up results to these results. But what I claimed earlier about inverse polynomial accuracy prohibits an FPTAS for the problem. But a PTAS is possible.

What we have is a quasi-PTAS. So if you have an n-th strategy game, and you're interested in some constant epsilon, then we can get you an epsilon Nash equilibrium, n to the log and over epsilon squared time. And because I'm looking at additive of epsilons, I'm also normalizing my payoffs to be 0, 1, so that epsilon is related to the maximum payoff of the game.

So that's what we have. And it's a great open problem if you can improve this, or show a lawyer bond. That's a great open problem. Speaking of which, if the rank of this game is up to, I believe, logarithmic, we know how to get a PTAS.

Now I don't have too much time left, so I guess I had two options for today. One was to talk about different problems that you can show PPAD hard, or to talk about other existence arguments. And I think I only have time for one of the two.

The other option is to show other arguments of existence in TFNP. So two examples, or other argument? What do you guys vote? So who wants two other examples? OK, who's giving the count? Who wants other existence arguments?

So I think there's a slight majority for other existence arguments, so I'm going to show that. PPAD is founded on the directed parity argument. Other natural arguments of existence in combinatorics are the following, and each of them is going to correspond to a different complexity class.

This is a parity argument on an undirected graph if an undirected graph has a node of odd degree then it must have another node of odd degree, also known as the handshaking lemma, I believe. Well this is another simple one. If a DAG, any DAG has a sink, that's going to give rise to the class PLS.

And that's the pigeonhole principle. If you have a function mapping n elements, n minus 1 elements, than there is a collision. That's going to give rise to the PPP. So PPA stands for polynomial parity on undirected graphs, PLS, polynomial local search, PPP, polynomial pigeonhole principle.

And I'm going to define them formally in the next few slides. So I'm going to start with PPA because it's very similar to be PPAD, except there's no direction. And the input to the problem is a circuit that induces a graph of [INAUDIBLE] strings.

So the circuit gets its input at nodes, and outputs two strings. Gets one string, outputs two strings. Now this circuit induces a undirected graph over all possible strings in the following way. There is a node between string 1 and string 2 if v1 is in the output list of v2, and v2 is in the output list of v1.

So this circuit gets n-bits as input, and has two n-bits as output. And no matter what it is, it doesn't use a graph over strings that places an edge between these two strings, if v1 is in the output list of v2, and vice versa. Now in the same reasoning as last time, the same spirit as last time, any input C is going to reduce a graph with a particular structure. What's that structure?

The odd degree of every node is what? At most 2. Hence the graph that is induced by any given circuit is going to be a collection of isolated vertices, cycles, and paths. The odd degree node problem is the following.

If the 0 string has odd degree, then I want you to find me another node with odd

degree, which we know exists by the handshaking lemma. Otherwise, if 0 to the n has even degree, just say, yes, and call it a day. Now PPA is the class of search problems in NP that are reducible, polytime reducible, to this problem, to the odd degree node problem, whose graph structure is very similar to PPAD, except there are no directions.

So any circuit C defines a graph of this form, except the graph is over exponentially many vertices, so we cannot just go and check every node, every node's degree. So the question is if 0 to the n has odd degree, then there must be another node of odd degree. Any of these nodes is a valid solution to the problem.

Something that I didn't mention before for PPAD, but it's useful to mention, and I'm also going to mention it for PPA, if I insist on finding the other end of this path, of the specific path that started at 0, the problem is not in FNP. So it's crucial that given the unbalanceness, or the odd degree of this node, I'll allow you to return any other odd degree node. That's very crucial.

If I insisted on you returning me the other endpoint on the path that starts at 0, that's above NP basically because there's no source certificate that this node in the other end of the path starting at 0. So it's very crucial that I allow you to return any odd degree node.

Now an interesting problem that is in this class, and as far as we know, not in PPAD, is the problem Smith, which is given a Hamiltonian circuit in a 3-regular graph, find me another Hamiltonian circuit in this graph. Now this graph is not exponentially big. It's actually a graph that you can write down.

So I give you a graph explicitly, and I also give you a Hamiltonian circuit in this graph. I claim that there is another Hamiltonian circuit. The question is find it. Now why is there always a Hamiltonian circuit?

Well, that follows from a theorem by Smith saying that-- a theorem by Smith implies that there's always another Hamiltonian path. And let me actually show it to you. It's very simple.

So here's is actually a copy from Papadimitriou's '94 paper. I guess there's a missing apostrophe here. So here's the input graph. And obviously, this is a Hamiltonian circuit. So I'm following the outer boundary here, going inside, and then coming back here. That's the Hamiltonian circuit.

And here's the same circuit, except I removed one edge, x and y. And here I'm showing a bunch of operations you can do this Hamiltonian path-- this is a Hamiltonian path-- to create another Hamilton path with the same edge missing. Hence, out of that edge, you get another Hamiltonian circuit.

And the [INAUDIBLE] is very simple. So what you do is the following. So you start with-- it doesn't matter. You can start with an arbitrary node. All the circuits I'm going to be defining, will have x as one of the two endpoints. But the other endpoint is going to be moving around and in the end, is going to get back to this vertex.

Hence, I can knock that edge back and get the circuit. So I started with the circuit. I remove one edge, x and y. And I'm about to show you that there is another Hamiltonian path that misses the same edge, but is different than this one. So it's going to be that one.

Now let me try to argue that that is another Hamiltonian path that is missing the same edge. How am I going to do that? Well, in this sequence of Hamiltonian paths that I'm going to define, x is going to stay fixed, y is going to be moving around. Now let's land somewhere in the middle, here.

So now y is this guy. y, because the graph is 3-regular, and y is an endpoint, there's exactly one edge that's used in the Hamiltonian path. And there are two edges that are missing. I'm going to try to add both of them and get a circuit-- and get a path, a Hamiltonian path.

What would happen if I tried to add this edge into this path? Well, that wouldn't be a path anymore. So because this guy would have degree 3 if I added that edge. Now have two kill one of the two edges adjacent to this node.

And I'm going to kill the one that the maintains the fact that the graph is connected. If I add this edge, and I kill this edge, I'm screwed. The graph is not connected anymore. So I'm going to kill that edge.

So what happens here? I added that edge, and I killed that edge. And this what I got. If I tried to add this edge, then this node will have degree 3. I have to kill either that edge or that edge. I can kill exactly one to avoid disconnecting my graph. And I'm going to be killing that one, and that's going to bring me here.

So every Hamiltonian path has exactly two neighbors corresponding to adding one of the two missing edges from the y endpoint of that path. And there's only one way this thing can stop, by arriving at y.

If I don't get endpoint y to come back to its original position I will keep going. In other words, like my proof [INAUDIBLE], what I did here is I defined a neighborhood relationship between Hamiltonian paths, which was adding an edge, one of the two missing edges from the endpoints. And another, the only stopping condition is if I get here.

And that's why Smith is in PPA. Now it's a very interesting problem to show that this is PPA complete. Let me define the class PLS. That was define earlier by Johnson, Papadimitriou and Yanakakis.

Now I want to implement a class that exploits this argument, that every DAG has a sink. The way I'm going to do that is I'm going to give you two functions. Function C is going to take-- both of them will have n-bits as inputs.

This guy, C, will output a list of strings, a list of k strings. So it has n-bits as input, and k n-bits as output. While this guy is outputting some real number, I mean, some rational number, whatever. But I interpret the output of the circuit as a number. I interpret the output of this circuit as a list of other strings.

I add that's between-- given these two circuits, that induces a DAG, I claim, in the following way. I add an edge from v1 to v2 if v2 is in the adjacency list of v1, and the score of v2 two is better than the score of v1. Then and only then, I add an edge

from v1 to v2.

And obviously, this is a DAG now because I'm increasing my score as I go along. So I cannot come back to where I started. So the problem that I want to define is the FindSink problem, which is given two circuits as above, find an x that has a better score than any of the adjacent vertices.

And such a thing has to exist because if I define this graph and find any sink of this graph, this will satisfy this property. The class PLS are all search problems in the P, but are polynomial time reducible to this problem, FindSink. The picture for this problem is this, exponentially large graph.

But there is a DAG that's implicitly defined by these two vertices, by these two circuits. And all of these are solutions. An interesting problem in this class is the LocalMaxCut problem, a relative of the well-known MaxCut problem, which is NP complete. In the LocalMaxCut problem, I'm giving you a weighted graph.

And I want a partition that's not globally optimal, but locally optimal, meaning there is no single node I could move from one to the other side of the cut to improve the cut value. Now if the weights were bounded and integral, this wouldn't be a hard problem.

But for arbitrary weights, it's actually PLS complete to find a local maximum cut. Final problem, and I'm concluding, is final class is the class PPP that's trying to implement the pigeonhole principle. Here I'm giving you a circuit that has n-bit input and n-bit outputs.

The collision problem that I want to define is given sets of circuits, either find me an x that maps to the 0 string, or find me a pair x and y that map to the same string. Clearly, by the pigeonhole principle if nobody goes to 0, there must be two guys that collide. So this problem is also total by the pigeonhole principal.

So it always has a solution, no matter what the circuit is. And the class PPP is all problems in NP that are reducible to this problem. Finally, the hierarchy of problems I defined is this. P, FNP, there is total FNP somewhere here, which I don't show.

And these are the relationships of these problems.

I haven't shown that these arrows true, that this is a subclass, PPD's a subclass of these two subclasses. This is easy. This is a simple exercise we can think about. This is basically my introduction to PPA, PPAD and related classes.

The final thing that I want to point out is answering a question that was asked after the previous lecture, which was why did you define these classes, and not just a TFNP complete problem. Why did you have to pay special attention to precise existence argument that gives rise to the guarantee that your problems are total?

And the reason for that is that actually TFNP is not what's called a syntactic class. In other words, if I give you a problem with TFNP-- if I give you a Turing machine, you cannot decide whether that is computing a total problem, that no matter what the input to that machine is, there's always an output.

So I had to pay attention to the specialized existence arguments because for specialized existence arguments, I know a priori that the problem is total. So in particular, no matter what circuit I give you here, I don't even have to check anything. No matter what input you give me, I know there is a solution.

No matter what pairs of circuits you give me here, I don't need to check anything. I know the answer to this problem-- there's always an answer to this problem, and so on, so forth. No matter what you give to me as input, it is important to define complexity classes for which you can show hardness results to find complete problems.

Otherwise, you would have what is called promise classes, which are not amenable to showing the completeness results. On that brief not, I want to stop. Thanks a lot.