

Computational Complexity
Prof. Subrahmanyam Kalyanasundaram
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad

Lecture -22
Ladners Theorem

(Refer Slide Time: 00:32)

SAT \in NP \setminus P. Is there a non NP-complete language that is in NP \setminus P?

NP-Intermediate

Ladner's Theorem: If $P \neq NP$, then there are languages ... not in P.

Hello and welcome to lecture 22 of the course computational complexity. This will be a rather brief lecture perhaps about 10-12 minutes and we will see what is called Ladner's theorem and we will see a high level outline of the of one of the proofs for the Ladner's theorem. So, we already know that P versus NP is a big problem it is a is a big open question we do not know whether P is equal to NP or P either P is a strict subset of NP. So, now the question is suppose P is not equal to NP what are the problems that are in NP but not NP?

So, we know that NP complete problems cannot be NP if P was not equal to NP because if NP complete problems was NP where NP then P would be equal to NP. For instance if SAT was shown to be NP then all of P is equal to NP all of NP is equal to P. So, for sure if P is not equal to NP then SAT and other NP complete problems like clique three colourability all of them are not NP. So, then the question is this. So, now let us take. So, suppose P is not equal to NP.

So, now let me draw a venn diagram suppose this is NP and suppose this is P the part that I am shading in blue suppose this is P this blue shaded part is NP. Now is it the case that. So, suppose this is NP-complete. So, maybe I will just write NPC sorry maybe I will just write NPC. So, NP is this whole thing this whole the entire thing the big circle here is NP NP. So, now suppose this is NP complete. So, this green shaded part is NP complete the question is are there.

So, we know that the NP complete languages are certainly not NP if P is not equal to NP. So, what I want to ask is are there languages in the middle or in the intermediate region. So, like in the light blue shaded region is this region empty or the are there languages that are in the intermediate zone. So, this part is called. So, NP is this whole thing and this middle part I am calling it NP intermediate.

So, given that P is not equal to NP we know that P and NP complete are disjoint. Now does NP constitute of only P and NP complete or is there an intermediate zone that is a question. So, is the blue is the light blue region in the middle are there languages there or are they empty this is the question. So, is there an in other words is there an NP complete language sorry is there a language that is not NP complete that is in in NP but not NP.

So, in other words is it a language that is in NP but not NP complete and not NP that is exactly what is called NP intermediate are there NP intermediate languages provided P is not equal to NP this is the question that is answered by Ladner's theorem.

(Refer Slide Time: 04:28)

NPTEL


Ladner's Theorem: If $P \neq NP$, then there are languages that are neither NP-Complete nor in P.
 NP: Intermediate languages.

Proof: Take SAT instances and use "padding".

$\phi = \overbrace{\hspace{2cm}}^{\hspace{2cm}}$
 $\leftarrow n \rightarrow$

$\phi \#^k = \overbrace{\hspace{2cm}}^{\hspace{2cm}} \# \# \dots \#$
 $\leftarrow n \rightarrow$

But the main trick is to figure out how much



So, what Ladner's theorem says is that if P is not equal to NP yes there are NP intermediate languages there are languages that are not neither in NP complete nor NP. So I will just give you a very brief high level idea because the proof is a bit technical and the techniques used are diagonalization mainly and we have seen already time hierarchy and space hierarchy where we use diagonalization.

So, I believe we could just uh. So, I will just give you a high level picture of this proof and there is an another trick used in this proof that trick I will explain in a bit more detail uh. So, that trick is called padding. So, what is sad SAT is an NP complete language where you given a Boolean formula you have to decide whether it is satisfiable or not. Now suppose I write ϕ , ϕ is a SAT instance.

And to write ϕ i require this much the formula takes this much length and usually in our usual terminology n is the length of the input and what we know is that in there is no polynomial time algorithm in the length of the input. So, if this is the input length let us call it n then there is no polynomial time algorithm in n that is; so, there is no n^k time algorithm for any constant k . Now what I can do is to like do something like this I could take a special symbol or I can even do it with some ones and zeros.

Maybe I can make a new special symbol let us say hash and I can add a lot of hashes maybe I will just write hash star. So the first part is exactly the same n of length n which is the same formula ϕ and the hash I will just change the hash to red color. So, that you know the distinction. So, I will just make it long just to; so, the blue part is just the formula ϕ and now there is a red part that is attached to the blue part.

(Refer Slide Time: 07:03)

NP. Intermediate languages

Proof: Take SAT instances and use "padding".

$\phi = \text{---} \xrightarrow{n}$

$\phi \# = \text{---} \xrightarrow{n} \text{---} \xrightarrow{n'} \#$

Suppose $n' = 2^n$. An algorithm that runs in $O(2^n)$ time is now a "linear time".

And now the whole length the length of this whole thing that means let us say this n prime. Now n prime is significantly bigger than n by because we added a lot of hashes to the end. So, in this picture it looks like maybe just maybe slightly more than two times bigger than n but think of n prime being much bigger than n . So, think of n prime being you can let us say you can add minim like much, much more not even a constant time maybe n prime could be exponentially longer than n .

So, in that case now if I have to decide on whether. So, there is a there is a Boolean formula here ϕ followed by a bunch of hashes now I have to decide if ϕ is satisfiable but now the problem length has grown so, big. So, the running time to decide this; so, if the running time of deciding SAT let us say there is an exponential time algorithm by just by trying out all the possible all the possible assignments.

So, here also we can do that but now that requires to power and time but when you measure the time as a . So, suppose n prime is suppose n prime is let us say 2^n . So, now what happens suppose if n prime is 2^n length. Now already now an algorithm that runs in 2^n power and time is linear in this. So, an algorithm runs in order to power n time you say is now a linear time algorithm it is just constant multiplied by input length because the input length is now.

So, much bigger than it is actually it is actually 2^n . So, or it is constant times n prime. So, what I am saying is that if n prime is sufficiently long then the now it is a linear time algorithm linear time means linear time is obviously polynomial time because it is just a linear factor. So by adding enough of these hashes now we can bring down the complexity of the problem because simply because the complexity is measured in terms of the input length.

And by blowing up the input length by not really increasing the problem size itself the actual problem is still only n size. So, by blowing up the input size input length but keeping the actually in the inherent problem size the same but the actual representation length by blowing that up we are now then measured as a function of the input length the problem becomes now linear time or polynomial time.

So, this is what is padding. So, by adding padding is just about by artificially increasing the length of the input. So that artificially increasing the length of the input so that the as a mesh when measured as a function of the input length the problem now becomes easier. So, from exponential time now we went to linear time which is polynomial time as well. So, this is padding.

So, you saw how to go from exponential to linear time now the same trick is used in the proof of Ladner's theorem as well. So, it takes SAT and adds a bunch of hashes to the end of SAT and tries to make it easy. So, the assumption again the starting assumption is that P is not equal to NP . So, SAT cannot be solved in polynomial time. So, that requires more than polynomial time it could be exponential or sub exponential something less than exponential also but certainly not polynomial time.

(Refer Slide Time: 11:40)

Now a known ...

But the main trick is to figure out how much to pad these SAT instances. We must pad enough so that the instances are not NP-complete, but not too much so that it becomes poly time decidable.

Candidates for NP-intermediate languages:

(i) GRAPH ISOMORPHISM:

The slide features the NPTEL logo in the top right corner. The main content is handwritten text in blue ink. A small video inset in the bottom right shows a man with glasses and a yellow shirt speaking.

So, now we want to pad SAT by enough number of hashes. So, that it no longer is NP complete the complexity comes down from the NP completeness. But we do not want to add padded. So, much that it becomes polynomial time. So, here we padded it. So, much so, that it became linear time. So, that is something we do not want we just want to make it not NP complete but we also do not want to do it. So, much that it becomes it brings comes down to P. So, that is a challenge.

So, you want to exit this green area but do not want to fall into the bottom dark blue area but want to remain somewhere in the middle light blue area. So, SAT starting from SAT we have to pad it enough. So, that it falls in the middle light blue area and this is what is proof is all about. So, it figures out how much exactly to pad by. So, that it comes out of NP complete but does not fall into P that is the idea of the proof.

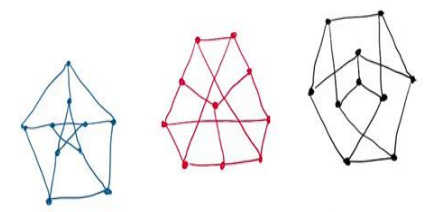
And it uses diagonalization as well amongst other things. So, the diagonalization is used to show that it is not NP. So, I will just give you some candidate problems which as of now we do not know whether they are NP and we also do not know that are not NP complete. So, most people believe that P is not equal to NP. So, people think that these are problems that are candidates to be the NP intermediate problems. So, I will explain the two problems one is graph isomorphism.

(Refer Slide Time: 13:26)


NPTEL

Candidates for NP-intermediate languages.

(1) GRAPH ISOMORPHISM:
 $\{ \langle G_1, G_2 \rangle \mid G_1 \text{ is isomorphic to } G_2 \}$



(2) FACTORING: $\{ \langle N, k \rangle \mid N \text{ has a prime } p \text{ s.t. } p \leq k \}$



So, what is isomorphism? Isomorphism is saying that these two are the same graph. So, it is just two graphs and whether these two graphs are the same. So, that is a bit abstract. So, I have an example. So, I have three graphs here the blue the red and the black in fact they are all the same graph they are all isomorphic to each other. So, let us see why they write isomorphic to each other. So, I will again i have not practiced this.

So, you can see the complexity involved. So, let me just label the blue graph 1, 2, 3, 4, 5, 6, 7, 8, 9, 10. So, isomorphism means that the second graph when we say that the blue and the red graph are isomorphic it means that the second and third graphs or sorry the blue and the red graphs are the same graph except for some rearrangement of the vertices and relay renaming. So, here I am giving it in the pictorial sense in a when you are giving it to a turing machine you may give the adjacency matrix or the adjacency list.

So, the matrix of the list the order of the vertices may have been jumbled up but just to make you understand I am just doing drawing a pictorial version of graphics amorphism. So, let us see. So, 1, 2, 3, 4, 5 forms a, 5 cycle. So, here we have to find a 5 cycle. So, maybe why not this; so, I will try to the claim is that let us say 1 2 3 4 5. So, these form a 5 cycle because I can let me try to outline these are the this form of 5 cycle and the remaining vertices also form a 5 cycle.

But then I have to be careful because one is adjacent to in the in the blue graph one is adjacent to 2, 5 and 6. So, the red graph it should also be adjacent to 2, 5 and 6 it is already adjacent to 2 and 5 now it should be also adjacent to 6. So, which makes this vertex 6 similarly 2 is adjacent to 1, 3 and 7 it is already adjacent to 1 and 3. So, it should this should be 7, 3 is adjacent to 8. Similarly so, this should be 8, 4 is adjacent to 9.

So, this should be nine and the central vertex should be 10 adjacent to 5. So, let us say let us see how the the inner in the blue graph how the inner inner star is connected 6 is. So, maybe I will just maybe I will just write down here in the inner star is 6, 8, 10, 7, 9 and back to 6. So, let us see whether the same thing is true here 6 well 6, 8, 7, 9 and back to 6. So, indeed it matches. So, this is why it is an isomorphism.

So, the it is the same graph but just redrawn in a different way and in fact even this graph is a redrawing of the same graph in a different way these 2 are both the same graph the red and blue what I am saying is that the black graph in the side that is also another isomorphic graph. So, that is the problem of graph isomorphism given two graphs we have to decide whether they are the same by rearranging the vertices can we get one from the other.

One from the other means the edges should be the same. So, here one is adjacent to 6, 2 and 5 that is the same in the blue graph as it is a red graph. And there are evidences. So, this is we do not know a polynomial time algorithm for graph isomorphism and there are strong evidences meaning there are some results that this is that indicate that this are not this is not NP complete meaning if this is shown to be NP complete something else happens which is which is unlikely.

So, I do not want to get into that but the point is that it is believed to be a problem that is not NP complete also we do not know a polynomial time algorithm. So, this is that is why people think this is an NP intermediate language.

(Refer Slide Time: 17:47)

6-8-10-7-9-6

NPTEL

(2) FACTORING: $\{ \langle N, k \rangle \mid N \text{ has a prime factor } d \leq k \}$

$\langle 61, 10 \rangle - \text{YES}$
 $\langle 101, 10 \rangle - \text{NO}$
 $\langle 121, 10 \rangle - \text{NO}$

The another problem that is believed to be NP intermediate is factoring. So factoring is the problem of given a number just to factor the number into prime factors. So, this is not a decision version of the problem but a decision version there are maybe more than one way to write a decision version one of such one of such a way to write a decision version is to write like this N, k where capital N has a prime factor of prime factor that is at most k . So, in other words we are asking does number n have a prime factor smaller than k .

So, for instance sorry I will use another color or maybe I will use another colour. So, for instance 51, 10 this is a YES instance because 51 has 3 as a prime factor which is less than 10. At the same time 101, 10 is a no instance because 101 is a prime number. So, it does not have a prime factor smaller than 10. In fact it does not have a prime factor smaller than 100 itself and even this is a no instance 121, 10 because 121 is simply 11 squared.

So, it does the smallest prime factors of is 11. So, it does not have a prime factor of size 10. Anyway the goal is you can certainly solve this decide this in polynomial time if you can factor in polynomial time but we do not know how to factor in polynomial time. So, this is also a problem in NP whether it is we do not know a polynomial time algorithm also it is not believed to be NP complete.

And maybe I will just say the same thing for graph isomorphism also given; so, it is the graph isomorphism is also an NP. Given a mapping it is easy to verify if I tell you put 1 to 7, 2 to 8 and such things like between the two graphs if I say in G_1 you can map it to G_2 and so on. Then you can verify that they are the same graph but so, it is an NP but there is no known polynomial time algorithm and also it is believed to be not NP complete same thing for factoring.

So, that is about NP intermediate languages and Ladner's theorem which simply states that if P is not equal to NP there are NP intermediate languages.