

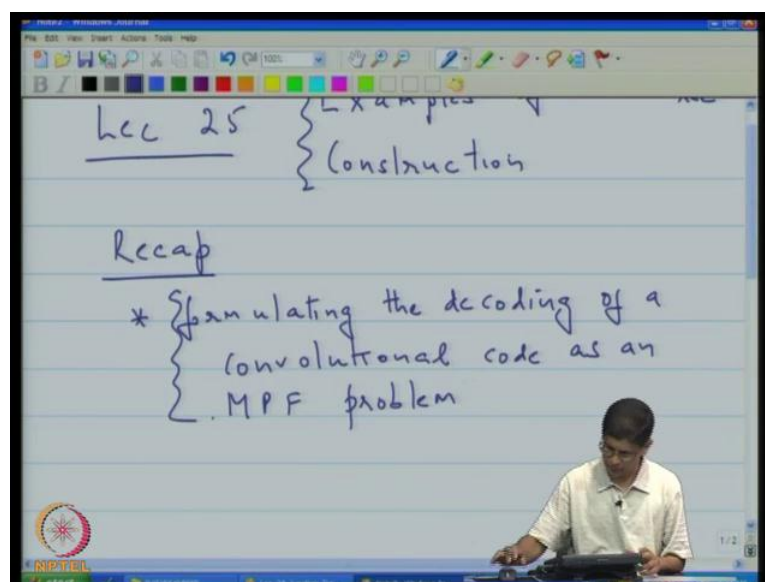
Error Correcting Codes
Prof. Dr. P Vijay Kumar
Department of Electrical Communication Engineering
Indian Institute of Science, Bangalore

Lecture No. # 25
Example of Junction Tree Construction

So, good afternoon and welcome, welcome back this is 25th lecture, and let me just as I usually do give you an update on what we been doing here. So, in the last lecture, we began by actually talking about, we began by formulating the problem of maximum likelihood code word decoding on a convolutional code as an MPF problem. And after that we said, now we know how to formulate problems as MPF problems; so how does one go about solving them? And we wish to solve the problem using this generalized distributive law approach, and the first step in actually applying the generalized distributive law is to actually construct something that is called a junction tree.

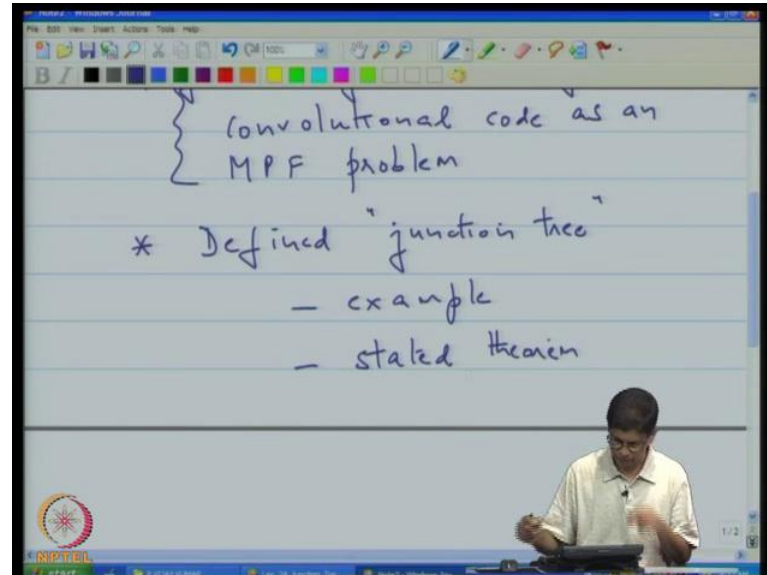
So, that later part of the lecture was devoted towards talking about junction tree. So, first we defined what a junction tree was, and then I showed you an example of a junction tree, and then the actual question is where, how do you construct a junction tree, and towards that I actually stated a theorem and will go ahead and start our lecture by proving that.

(Refer Slide Time: 01:56)



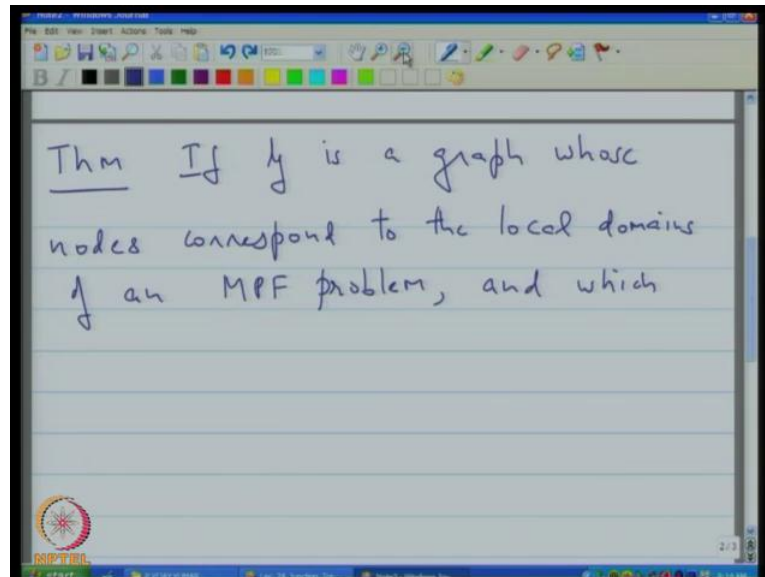
So, this will be our lecture 25 and what this lecture will be about we will be talking about junction trees and message passing; so examples of junction tree construction. So, as a quick recap of our last lecture, we began by computing and discussion on formulating formulating the decoding of a convolutional code; code as an MPF problem.

(Refer Slide Time: 03:40)



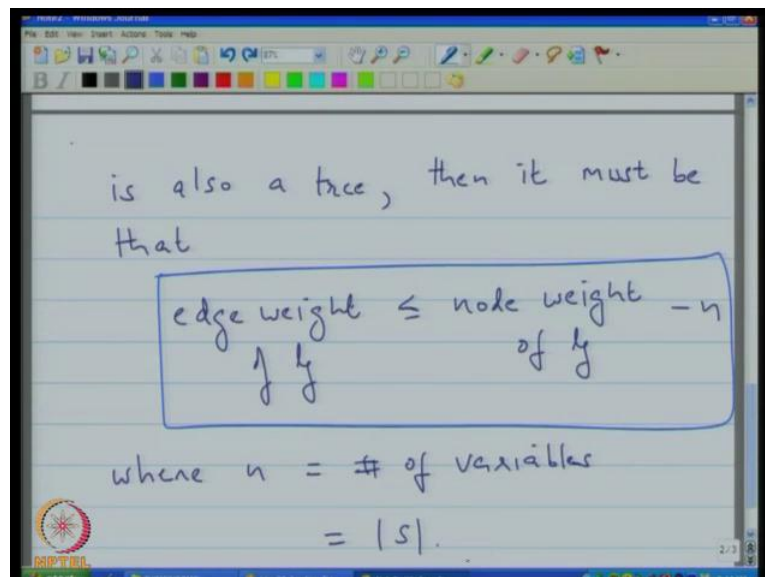
And after that I defined what it means to be a junction tree, we looked at an example and then a stated theorem and today our task is actually to prove theorem. So, towards that let me do the following that may actually reproduce the theorem on the current lecture.

(Refer Slide Time: 05:02)



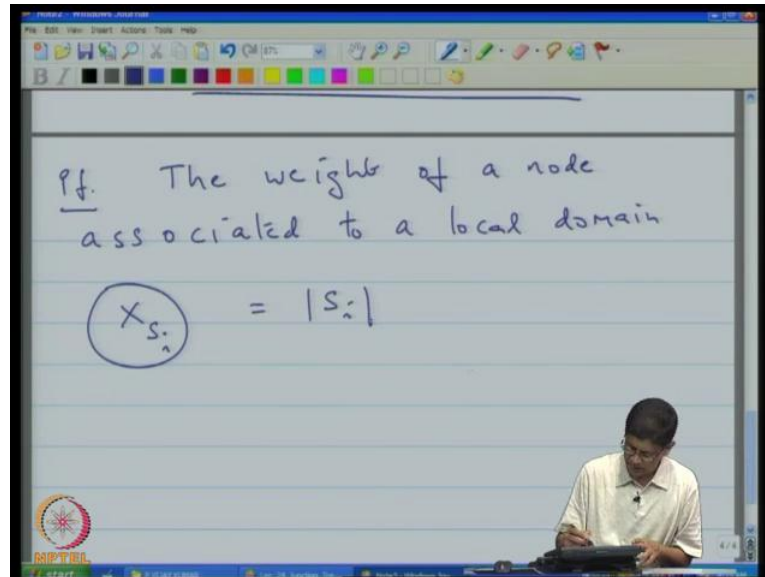
So, that in work to well, so I think I am going to have introduce a new page for that either copying then I attempting to do here. So, let us paste that on this page so here we go so let me just I just drag this down here. So, if g is a graph whose nodes correspond to the local domains of an MPF problem.

(Refer Slide Time: 06:14)



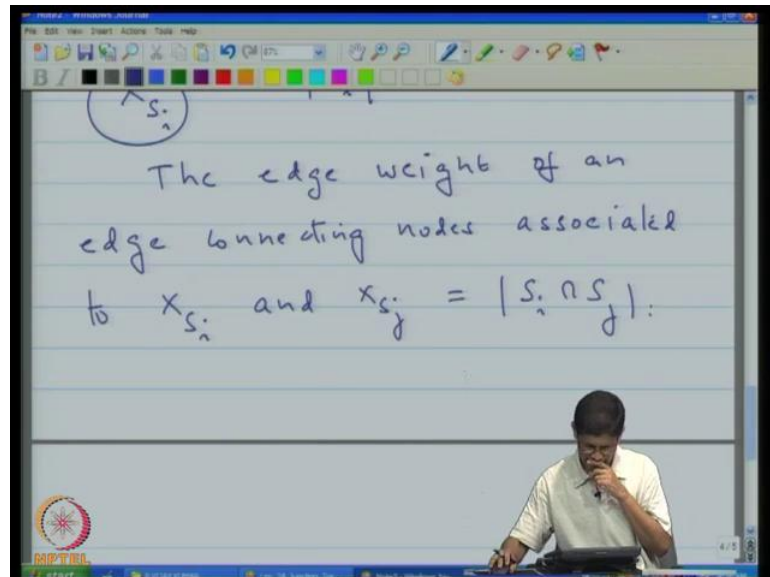
And which is also a tree, then it must be that the edge weight must be less than or equal to the node weight of the graph minus n , where n is the number of variables, and that is also the size of the universal set and the proof is actually quite simple.

(Refer Slide Time: 06: 33)



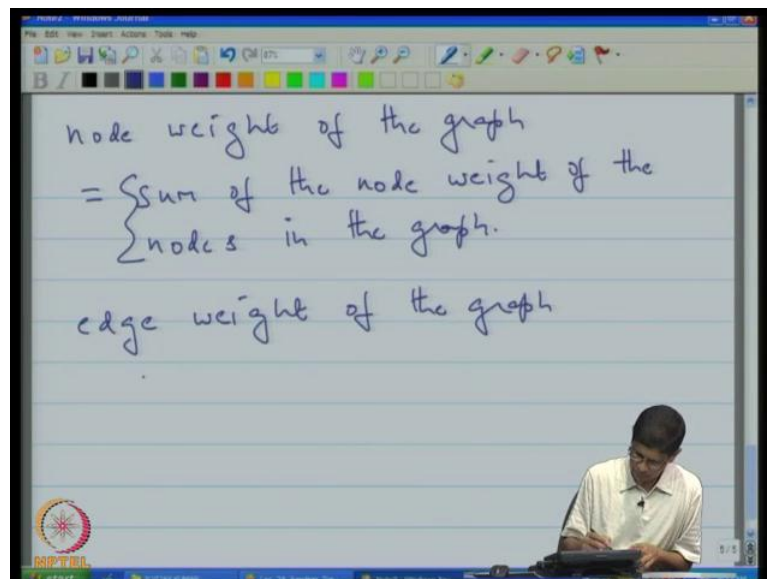
So, the proof is let let I believe. So, let us I guess you have to clarify what we mean by the node weight. The weight the weight of a node associated to a local domain x_{s_i} is equal to the size of the set s_i , on the other hand the weight the edge weight.

(Refer Slide Time: 07:47)



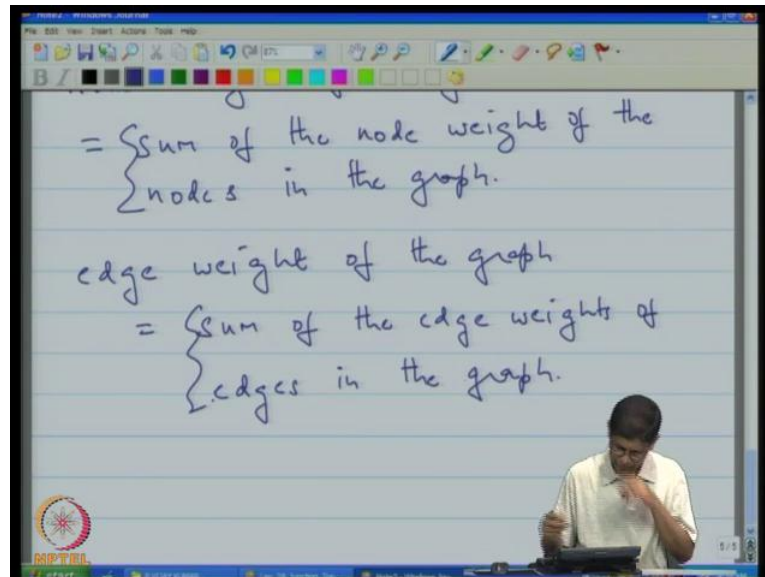
The edge weight of an edge connecting nodes associated to x_{s_i} and x_{s_j} is equal to the size of the intersection.

(Refer Slide Time: 08:48)



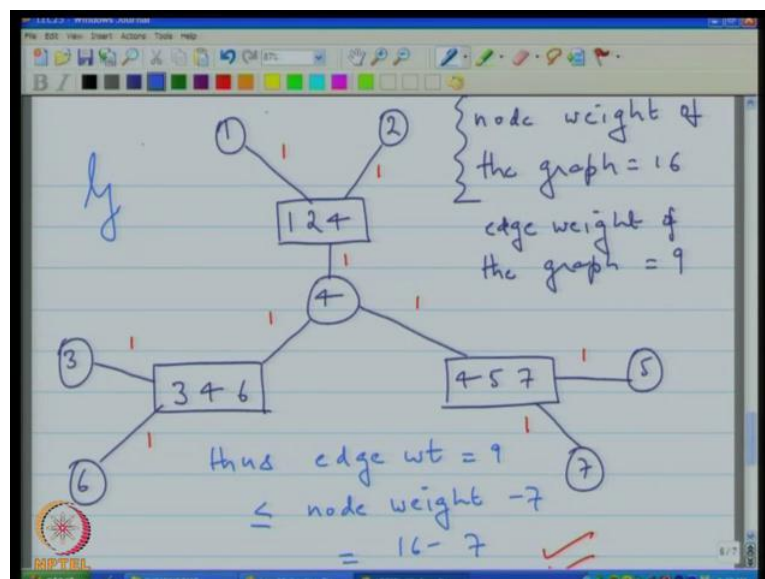
And the node weight of a graph of the graph is equal to the sum of the node weight of the nodes in the graph, and then the edge weight of the graph is equal to in a similar way, it is equal to the sum of the edge weights of edges in the graph.

(Refer Slide Time: 09:40)



So, perhaps we should look with an example. So, I am going to use the same example we had last time.

(Refer Slide Time: 10:31)



So, in this graph you actually see that there are total of ten nodes and these nodes over here have node weight one because there associated with a single variable these have node weight three so from this it follows that the node weight of this graph, node weight of the

graph is equal to ten; how about the edge weight to look at the edge weight, what we have to do as we have to assign weight to the edges.

So, when where assigning weight let us say it to the edge connecting these two nodes that weight is one because the size of the intersection of these two sets is one similarly, it is one here. So, in fact it turns that for every edge here the intersection actually equal to one. So, let us put that down, so we will put down a weight of one, one, one, one, one, one, one, one and one. So, from this we get that the edge weight of the graph, of the graph is equal to let us add, so 1, 2, 3, 4, 5, 6, 7, 8, 9.

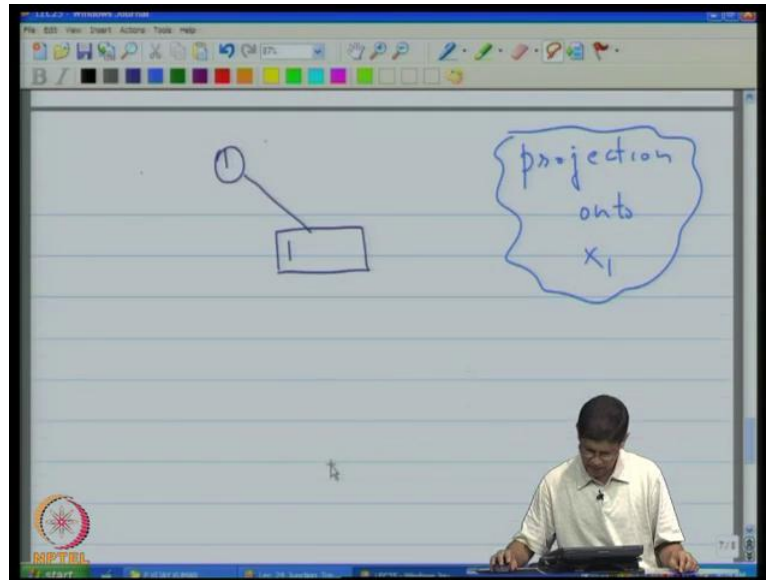
So, there actually excuse me this never here, the node weight is actually there are seven singleton nodes for a total of seven that then this three, three, three. So, that should be 7 plus 3 times 3 that should have been 16 sorry about that the edge weight on the other hand we did complete correctly, 1, 2, 3, 4, 5, 6, 7, 8, 9. So, the edge weight of the graph is actually equal to 9. And what are theorem actually said theorem that we trying to actually prove now, it says that the edge weight of of a graph associated to the local domain given there it is a tree must be less than or equal to the node weight minus the number of variables.

So, in this particular case we see that the difference between the edge weight and the edge weight is less than or equal to 16 minus 7. So, since the number of variables is 7 the theorem holds in this case. So, thus thus the node weight the sorry the edge weight thus the edge weight, which is equal to 9 is less than or equal to the node weight minus 7 because this is 16 minus 7. So, we see that the theorem actually is holds in this case, but we still have to prove it now, for the other part of the graph. So, let us one of the small piece of notation, let us actually call this entire graph graph g .

So, this is a tree and it is a tree where the vertices are in one to one correspondence with the local domains that arrest from the MPF problem. Now, I would like to go back to our earlier lecture and in which we actually talked about the projection of this graph, we shall recall, we said that has a graph one weight to test whether or not a graph is a junction tree given , that is already is a tree is to actually take the graph and project it on to each of its individual variables and of the graph that you get in that manner is in each case a tree, then you have

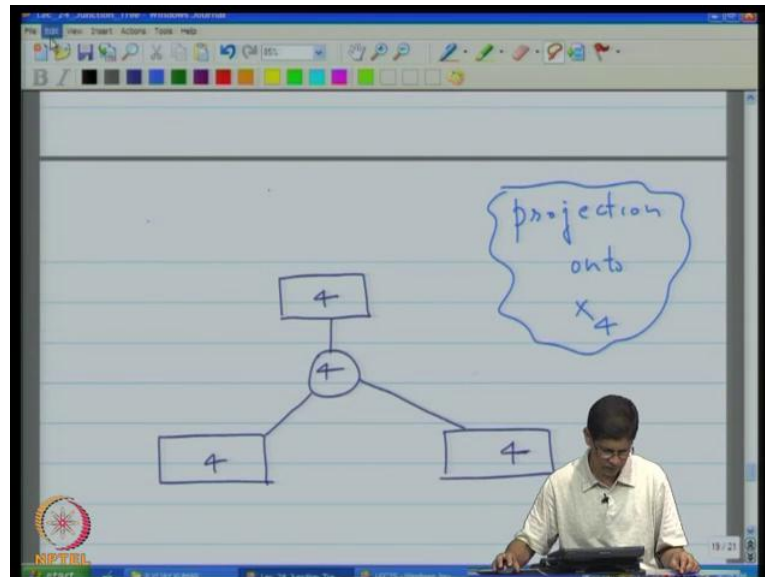
that what you had recently is a junction tree; that was one of the two equivalent definitions of a junction tree.

(Refer Slide Time: 15:27)



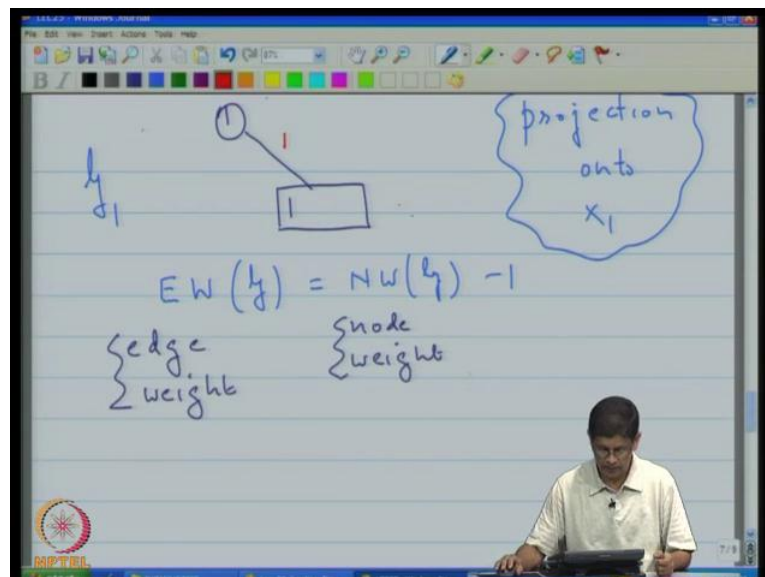
So, when we projected this graph, and here is the projection on to x_1 , you see these two nodes, and we see that this is a tree. So, in our notation what we are going to do is we are going to call this in get respect we are going to call this the graph g_1 . So, since I want to make use of this let us again select a page and copy it on to our current lecture, and I am also going to use the next projection over here.

(Refer Slide Time: 16:01)



So, let me select that is well and copy that is well good. So, we have both of this copied and now, I am going to term this since this is the graph obtained by projecting on to x_1 .

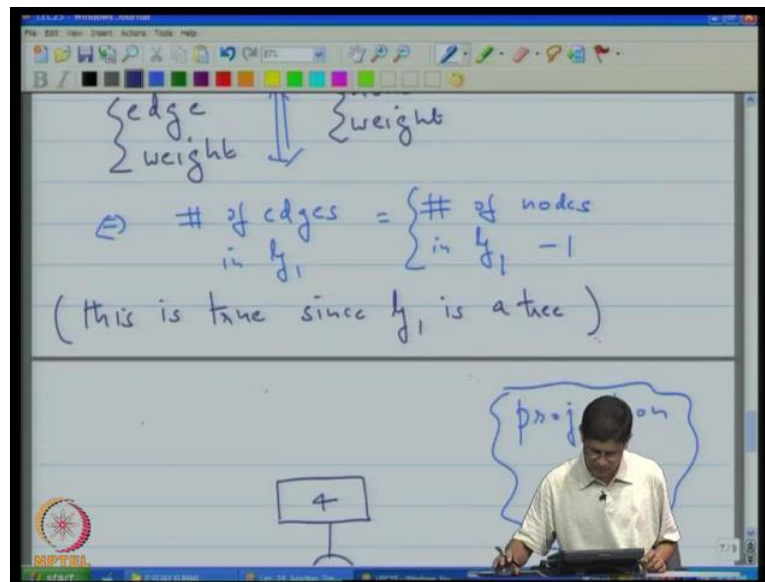
(Refer Slide Time: 16:22)



I am going to call this g_1 and you will actually notice that the edge weight of g_1 is equal to the node weight of g_1 minus 1 well where is that well, first of all I guess it is clear that by EW I mean edge weight. And similarly, that by NW I mean node weight all right and if you

were given that this is a tree, then this is true because in these projections every vertex is label with just a single variable and it and the variable is common to all the vertices in the projection. So, the edge weight is also equal to 1. So, for example, so when you actually computing this all that you are doing is really, this is saying the number of edges is equal to the number of nodes minus one, but that always holds for a tree.

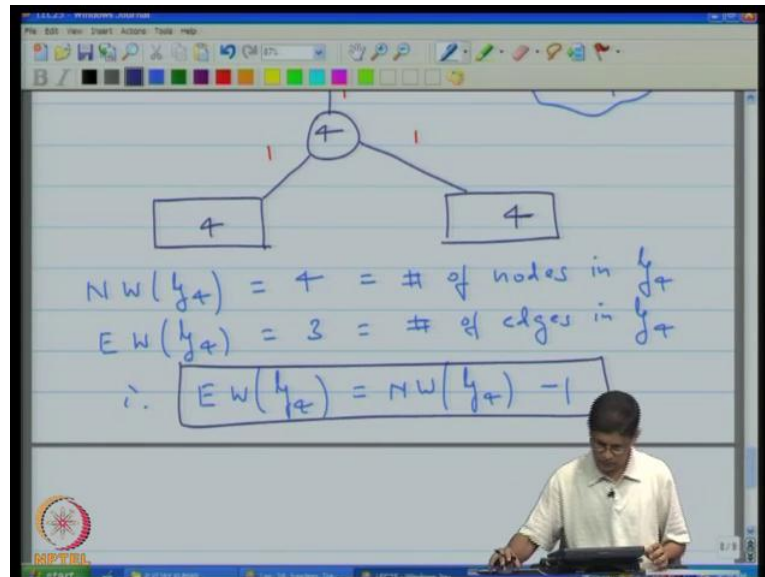
(Refer Slide Time: 17:38)



So, we just to point out that this equivalent in its projection graphs to actually saying. So, this is equivalent to saying that the number of edges in g_1 is equal to the number of nodes in g_1 minus 1 and this is true this is true, since g_1 is a tree. Now, let us look at the second projection here. So, this is the projection of the graph that we have over here onto onto the variable 4.

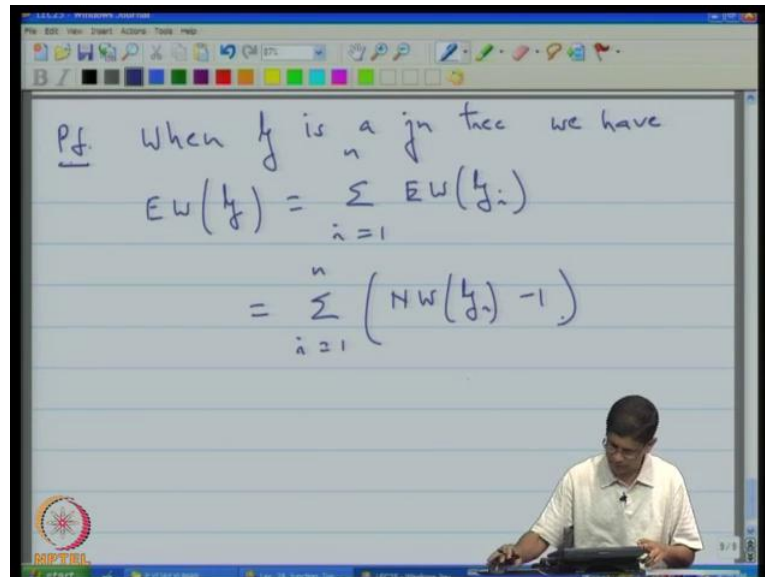
So, when you project this onto 4 what you do is you retain only this vertices, which have contained 4 reject other vertices and retain all edges that connect to vertices that contain 4. If you do that you get the graph that I just showed to you and then even the other minor point is that, even when we label with regard to labels with graph all variables other than 4. So, again once again we have the property that all the vertices have single variable so that node weight is each equal to one and all the edge weights are actually equal to 1.

(Refer Slide Time: 19:06)



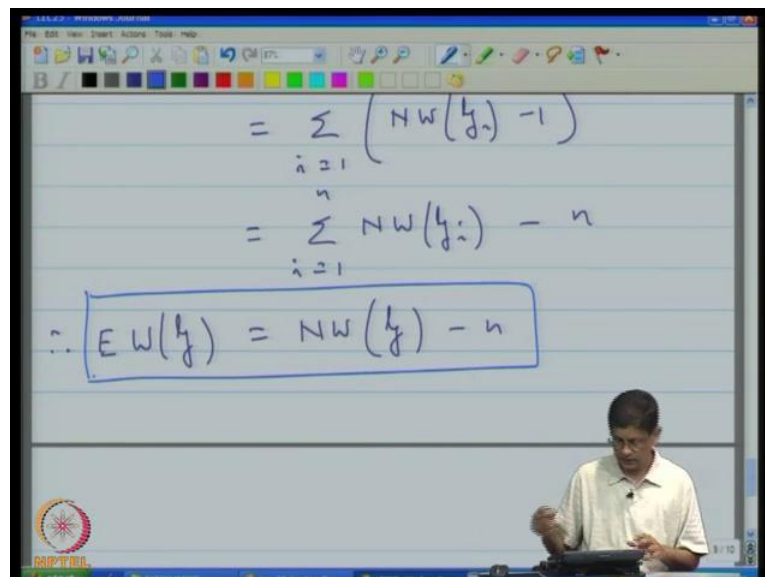
So, once again you have in this case that the node weight of g_4 , the projection on to x_4 is equal to 4 which is equal to the number of nodes in g_4 , the edge weight of g_4 is equal to 3 is equal to the number of edges in g_4 . Therefore, it is once again true that the edge weight of g_4 is equal to the node weight of g_4 minus 1. Now, let us get back to our proof so our theorem and our goal is to actually show that, if g is a graph whose nodes correspond to the local domains of an M P F problem and given that the graph is also a tree then it must be true that the edge weight of g , which is the sum of all the weights of all the edges is less than or equal to the node weight of g minus m and the proof is so with this information or proof goes as follows.

(Refer Slide Time: 20:34)



We say that since see when when g is a junction tree, we have as we just seen in the examples that the edge weight of g is the sum of the edge weights of g_i and since it is a junction tree each of these projections is a tree. So, this is the sum i is equal to one to n node weight of g_i minus 1, which is by definition equal to the node weight.

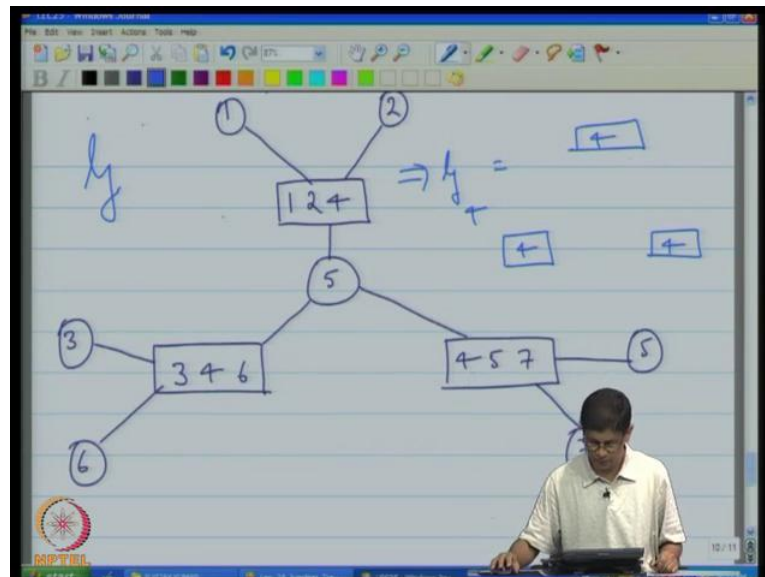
(Refer Slide Time: 21:28)



Let me put on intermediate step here so this is the sum i is equal to 1 to n of the node weight of g_i minus n , which is equal to the node weight of g minus n ; therefore, we have that the edge weight of g is equal to the node weight of g minus n provided g is a junction tree. Now, so what happens what would happen, if let us say g were not junction tree.

What would happen is that if for instance this was not a junction tree then what you would have is that when you project on to each of the variables, you would get a graph that is not a tree, in the sense that it would not be connected and so it in some sense you can equate as a union of trees, which in graph terminology is called a forest. So, perhaps, I can do let us see if I should let me right, I think that I will take this example and approx is the best way to state with second example.

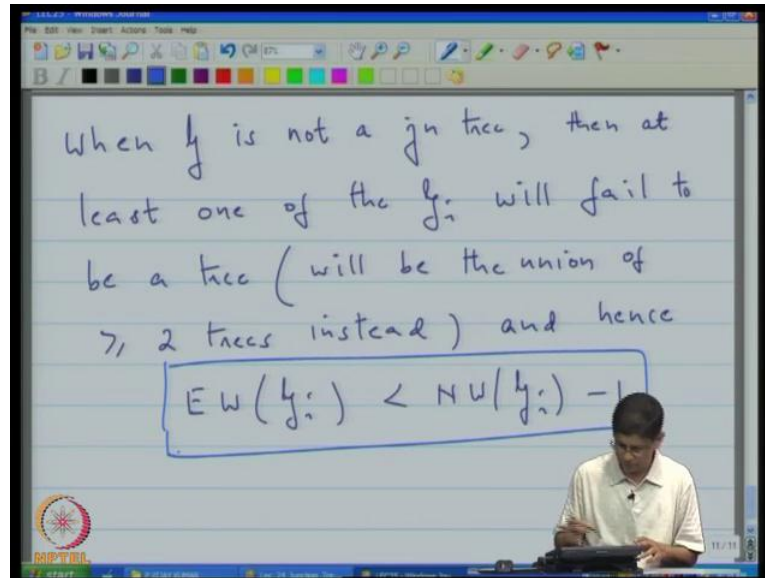
(Refer Slide Time: 24:02)



Let me delete all the material that I do not need and let us say that in a certain instance for example, we had that this thing over here was not a 4 that let us say that it was instead a 5. Supposing, this was our graph can a certain instance then when you project onto let us say you project onto 4. So, in this case g_4 would look like 4 corresponding to this node. Then there would be 4, which corresponds to this node here and let be a 4 that corresponds to this node over here. So, what you would have is not a tree, but in general union of tree, as trees are of forest within each tree its actually true that the number of nodes exceeds the number

of edges by one, but when you add overall the trees the deficit is no longer one, but equal to the number of trees.

(Refer Slide Time: 25:25)



So, for this reason we can say that when g is not a junction tree then at least one of the g_i will fail to be a tree will actually be a forest, will be the union of trees of greater than or equal to two trees instead and hence in this case, what will happen is that the edge weight of this g_i will be less than the node weight of g_i minus 1 in other words, you will not equality will not hold.

So, will have strict inequality so then when you so as a result of this as a result of this when you go up here, and try to do this same argument that you did for the case, when g was a junction tree and you sum over such this edge weights then you will actually, then you want to have equality here, you will have in equality here which means that the edge weight is less than the node weight minus n . In this case, so let us give the equation that we had up earlier a number.

(Refer Slide Time: 27:36)

> 2 trees instead) and hence

$$EW(y_i) < NW(y_i) - 1$$

If we now proceeded to argue as when deriving (1) we would end up with

So, let us call this one equation one and we would find if we now proceeded to argue as when, deriving 1 we would find that or we would end up with we would end up with the edge weight of g be less than the node weight of g minus the number of variables. So, we would actually have this.

(Refer Slide Time: 28:10)

deriving (1) we would end up with

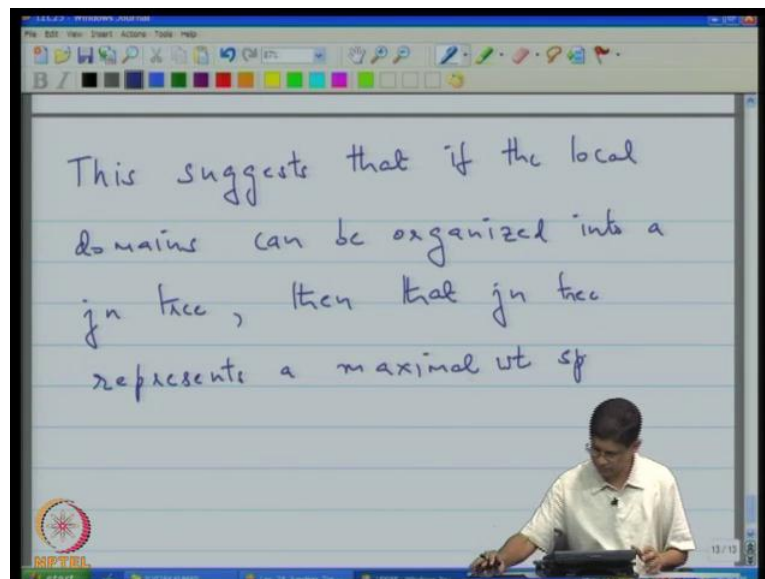
$$EW(y) < NW(y) - n$$

the theorem follows.

The theorem now follows and the reason, the theorem follows is because we have already shown that the edge weight is equal to the node weight minus the number of variables. When it is a junction tree and if it is not a junction tree, we have strict inequality. So, that proves that the largest value that the edge weight of g can possibly be is the node weight of g minus n and when you achieve that equality then what you have is a junction tree.

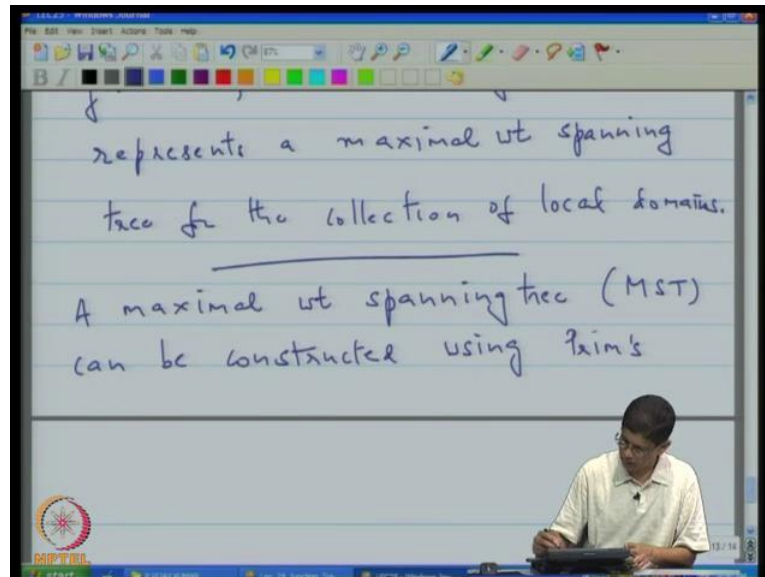
So, the lesson to take away from this is that, well it looks like you are trying to maximize this edge weight and when you are constructing this graph, why do not we try to find a graph. So, you have a set of local domains, why do not you try to construct a graph out of it which is a tree and there which which connects all the nodes. So, it must be what is called a spanning tree and it must have maximal edge weight.

(Refer Slide Time: 29:45)



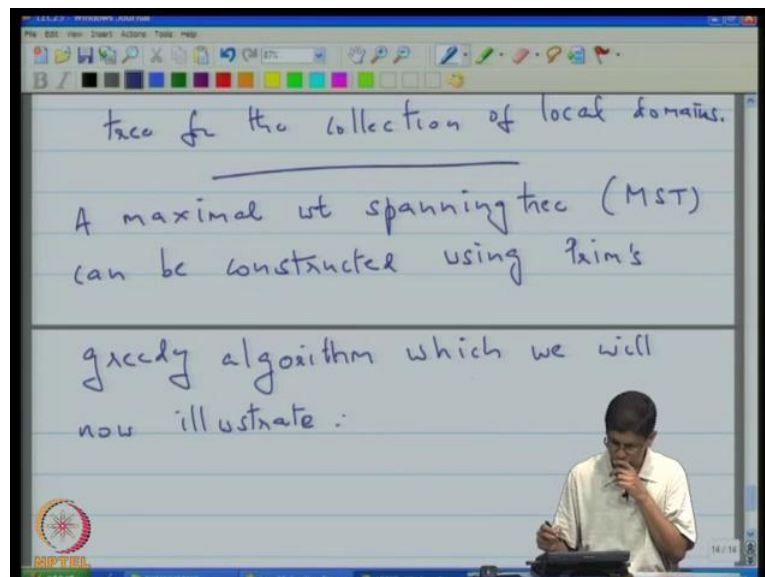
So, that is sort of all this is that this suggests this suggests that, if the local domains can be organized into junction tree, then that junction tree represents a maximal weight spanning tree for the collection of local domains.

(Refer Slide Time: 31:16)



Now, in the computer science literature our people are there are two well-known algorithms for constructing minimal weight spanning trees, these are easily adapted to constructing maximal weight spanning trees.

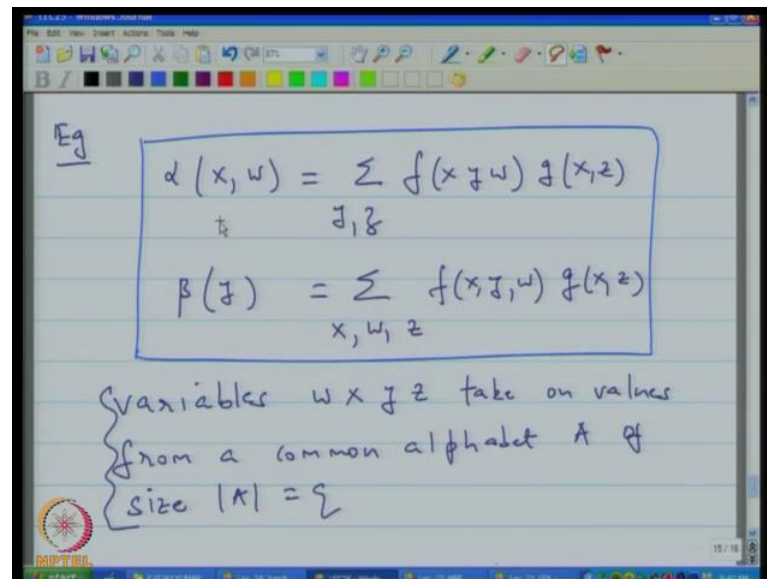
(Refer Slide Time: 32:34)



So, the algorithm is people commonly talk about are Prim's greedy algorithm as well as and other second algorithm due to crush cal. However, will follow Prim's greedy algorithm and I

want actually describe formally the algorithm. Since, I think it would consume perhaps more time then we have what I will do however just illustrated in the case of some examples a maximal weight spanning tree and will abbreviate this by M S T can be constructed using Prim's greedy algorithm, which we will now illustrate. So, let us start with an algorithm, which we came across some time back and which has to produce here.

(Refer Slide Time: 33:35)



Eg

$$\alpha(x, w) = \sum_{j, z} f(x, j, w) g(x, z)$$

$$\beta(j) = \sum_{x, w, z} f(x, j, w) g(x, z)$$

variables w, x, j, z take on values from a common alphabet A of size $|A| = 9$

So, this example taken from lecture, four lecture ago what we would interested in that time was computing two different functions, and you can think of this is as example of MPF problem in the obvious way, one difference from pairs some of the other example we look at is that there are two objective function in this case.

(Refer Slide Time: 34:17)

$$d(x, w) = \sum_{j, z} f(x, j, w) g(x, z)$$

$$\beta(j) = \sum_{x, w, z} f(x, j, w) g(x, z)$$

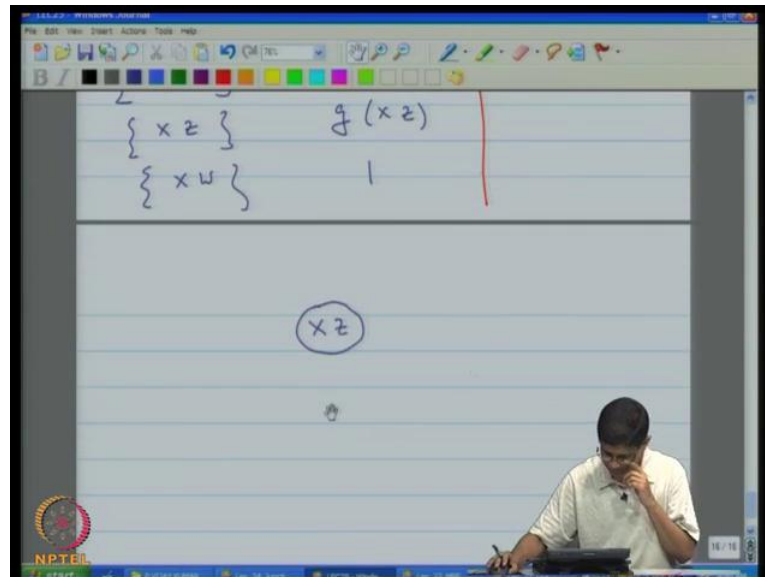
objective functions

LD	LK	LD	LK
$\{x, j, w\}$	$f(x, j, w)$	$\{j\}$	1
$\{x, z\}$	$g(x, z)$		
$\{x, w\}$	1		

So, these are two objective functions. So, based on this we can make a least for local domains so our local domains in this case are x, y, w which is associated to local kernel f of x, y and w then you have x excuse me, then you have x and z associated to g of x, z and then you have looking at objective function.

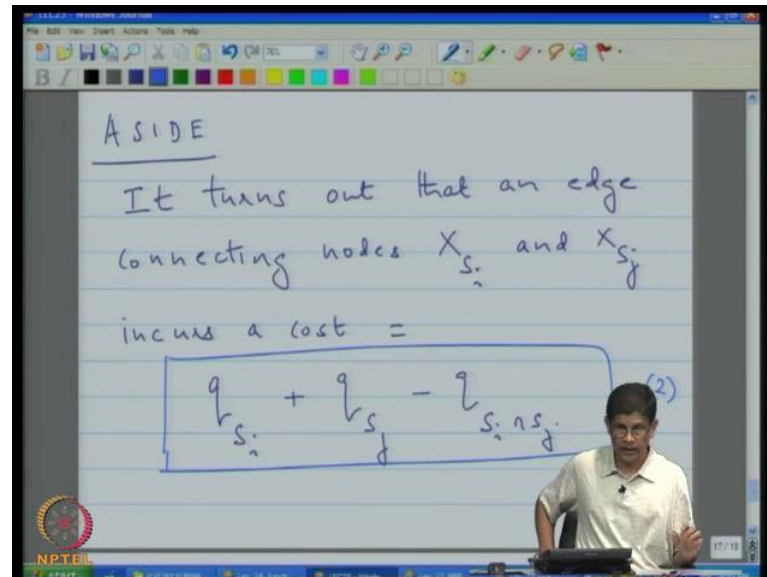
We have x, w and y so you have x, w associated to since this is an objective function what we do in the cases just sign kind of a dummy local kernel which is one and finally we have just one more, which is we have local domain y . Once again since there raises from objective function, which is associated with dummy local kernel one. In this case we have this four local kernels these two objective functions the global kernel of course, is the product all of these, which is the product of these two now, we want to apply Prim's greedy algorithm to actually organize this local domains into junction tree. So, one there are 1, 2. 1, 2, 3, 4 local domains we can pick any one of them and actually start this process.

(Refer Slide Time: 36:31)



So, let say what we pick x, z just like that. So, we take one of the local domain then we actually tie up a node now our next past is actually took take in Prim's greedy algorithm says well, try to maximize your edge weight and every opportunity. So, that means that now you want to connect to one of these, you want to connect one, two, three one of these nodes you cannot repeat excide of course, one of the three remaining nodes well maximizing the edge weight well the edge weight is the is the number of variables that lying in the intersection. So, you can actually connect this, because x, z in these haven intersection of one or we can connect this which is also on intersection of one that it terms out that it terms out that.

(Refer Slide Time: 37:23)



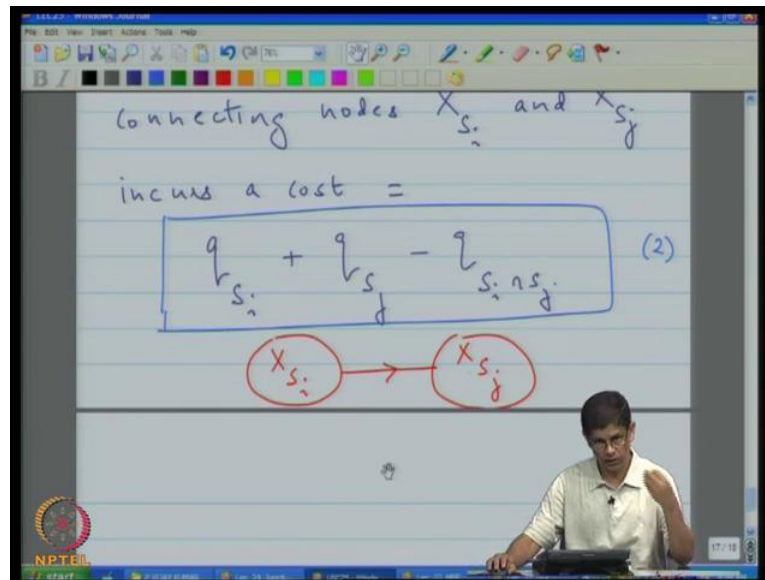
So, I just make on side comment here a side it turns out or items out that an edge connecting nodes X_{s_i} and X_{s_j} incurs a cost equal to an equal to q of s_i plus q of s_j minus q of $s_i \cap s_j$. So, let us call this equation two. Now, I should point out that. So, what I am actually saying is that given a bunch of local domains you want to actually construct a local domains and I pointed out that, if to in a junction tree and if junction tree can be constructed then that junction tree will correspond to a maximal weight spanning tree.

So, let is instead apply an algorithm that gives a maximal weight spanning tree and where going to use Prim's greedy algorithm and Prim's so greedy algorithms means that, you look for instance clarification that you try to actually maximize your current profit you do not think long term. So, that means that whenever you have given that you have in existing node you are going to connect try to another node associated with maximum possible edge weight.

So, that is are primary consideration that, but sometimes has in this particular case we might face with two options there is given that we started out with x, z and we want to apply a greedy algorithm, we could even a either join x, z , either x, y , either w or x, w and in such cases, then you can actually look little beyond Prim's algorithm that is go by and beyond what you need construct a junction tree and start thinking about this criterion, it turns out the

computational complexity of having in your graph two nodes that are like this, which are connected with an edge in other words.

(Refer Slide Time: 40:08)



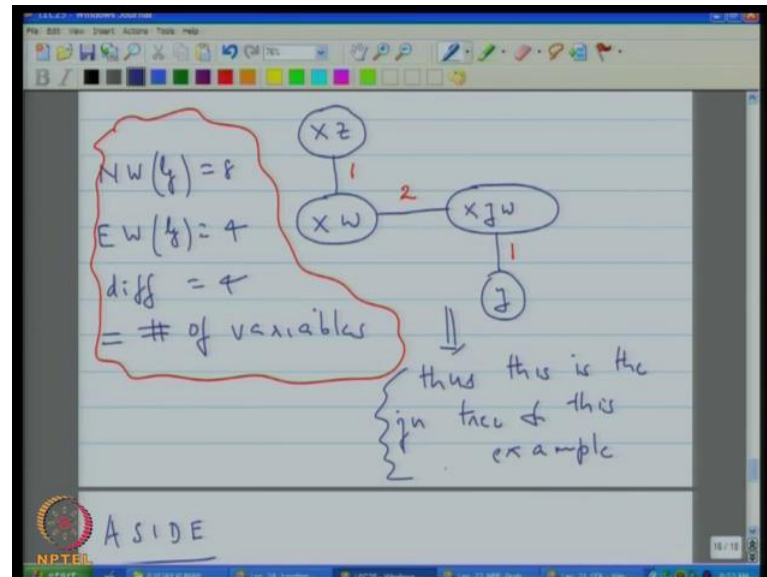
If you have a graph like this which has x_{s_i} , x_{s_j} and these are connected by an edge then we turns out that the cost of having there is an edge cost to having an edge like this in your graph because after all your computing, you are going to solve the M P F problem by computing the objective function. So, that involves the computation complexity, so that cost majored in number of operations the operation being the multiplication or addition it turns out that cost in turns out number of operation that having these two nodes connected in your graph is the size of the alphabet of the variables in a side taken to gather.

So for example, if I consist of two variables each of which is drawn from an alphabet of size q then the size of alphabet of pair of variables together with q times q square. So, that sense that we mean this, so the cost of having this particular segment of graph is q of s_i plus q of s_j minus q of the intersection size of intersection. So, that means when you actually come down here in say a work if I actually put this all incurred cost of x square let us assume that all the variables are drawn. I think in our example all of them when drawn from an alphabet of size q . So, a parts just I am precise it let me repeat that here.

(Refer Slide Time: 41:45)

So w, x, y, z all are drawn from A and size of A is equal to q this was our setting the set only the commons here. So, given that and the sign going to then therefore, choose to actually connected to xw because I know that the cost incurred will be less so make this connection. So, there are total of four local domains and I need to connect make this as spanning tree. So, I need to have a tree, I need to run all through all the edges and I want to maximize by edge weight. So, here my edge weight here turned out be one and now I have y and I have xw . So, Prim's algorithm says where I can either put xy , I can either connected to this out of this, but then if I connect xw to this all maximize my edge weight it will become two.

(Refer Slide Time: 43:03)



So, I am going to do that so prim will say in this case which actually put x y w drawn here, and now and now would I actually have is an edge whose weight is two, which is one local domain left which is y and again it is clear that you must connect y here, you must connect y here so will do that and corresponding edge weight is one. So, in this particular case, what we have is that you can actually check there small side calculation to check that the node weight of all graph.

The graph that we actually constructed in this weight is eight that sum of the node weights of each of the nodes and the edge weight is actually equal to four, and the difference is equal to four, which is equal to the number of variables. So, that proves this is the junction tree of course, it also proves that maximum weight spanning tree because the maximum possible value of the edge weight of this graph was in the node weight minus number of variables. The node weight is six because it is sum sizes of the local domains.

So, the only think that was the variables were edge weight and you may it at largest possible. So, this is the junction tree for this example, thus this is the junction tree for this example. Now, let us next look at second example. So, let us go ahead in call this we also call this example one. So, this is example one for example two, let us go back to the Walsh transform computation and which I guess perhaps I should see if I can quickly bring that down is well.

(Refer Slide Time: 47:16)

Fig 1 (The 8-dimensional Walsh-Hadamard Transform)

$$F(x_4, x_5, x_6) = \sum_{x_1, x_2, x_3} f(x_1, x_2, x_3) (-1)^{x_1 x_4 + x_2 x_5 + x_3 x_6}$$

So, here is our example 2, now so perhaps I can actually delete this page here. So, this is our example two in which we have the Walsh transform and you can actually see the local domains and local kernels way easily, but it will put that down again. So, the local domains actually will be computed, so perhaps we can put this down as well need not. So, I am going to repeat never mind, so I will repeat local domain.

(Refer Slide Time: 48:01)

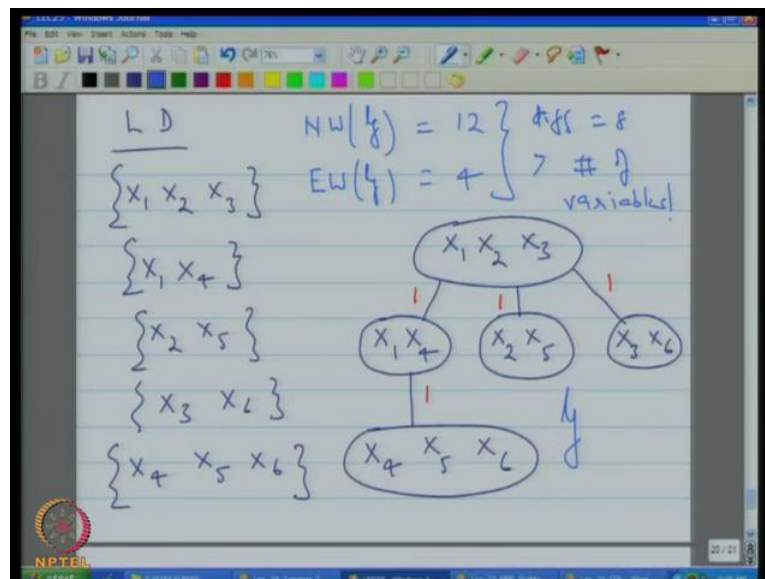
LD	LK
$\{x_1, x_2, x_3\}$	$f(x_1, x_2, x_3)$
$\{x_1, x_4\}$	$(-1)^{x_1 x_4}$
$\{x_2, x_5\}$	$(-1)^{x_2 x_5}$
$\{x_3, x_6\}$	$(-1)^{x_3 x_6}$
$\{x_4, x_5, x_6\}$	$F(x_4, x_5, x_6)$

So, the local domains are just for this computation, we have the local domain $x_1 \times x_2 \times x_3$ where $x_1 \times x_4 \times x_2 \times x_5 \times x_3 \times x_6$ and then $x_4 \times x_5 \times x_6$ so we have a total of 1, 2, 3, 4, 5 we have a total of 5 domains. So, I like this put this down, so we have $x_1 \times x_2 \times x_3$ and you have $x_1 \times x_4$. we have $x_2 \times x_5$ and have $x_3 \times x_6$ and you have $x_4 \times x_5 \times x_6$, the corresponding local kernels are although there not very important in this case, but just keep track of thinks $x_1 \times x_2 \times x_3$ minus 1 to the $x_1 \times x_4$ minus 1 to the $x_2 \times x_5$ minus 1 to the $x_3 \times x_6$ and f of $x_4 \times x_5 \times x_6$.

So, the reason actually said that the local kernels are not important because they do not coming to the picture until after you construct a junction tree. So, when you are attempting to construct a junction tree you are going to work only with local domains. So, in this case we have 1, 2, 3, 4, 5 and we want to construct a local domain. So, let us that Prim's greedy algorithm so for that would I am actually going to do is I am going to try to construct along a side this local domains.

So, let us copy this page again here we go and now I am going to delete this we should only well try to construct a graph on this side. So, supposing I started does not earlier my Prim's algorithm and as we start from which given node to pick. So, let say I start with $x_1 \times x_2$ and x_3 .

(Refer Slide Time: 51:04)



Start with let us put that down here. So, $x_1 \times x_2 \times x_3$ so that is my first node and now when I see that we maximal edge weight, if I happen any of this I cannot happen this because edge weight is zero, for any one of this edge weight is one. So, I am just going to pick any one of this random. So, let us say there I put an excuse me let me put $x_1 \times x_4$ over here and I connect this and edge weight happens to be one since intersection is just one.

Now, again I look at this and I say well I now can connect, I have again I have two choices here and even if I would apply this other criterion of actually minimizing the number of computation q of s_i plus s_j minus, this it is not going to resolve it is not going to differentiate between these, because for example, I have a choice of depending this over this to this. Now, if I upend this to this the edge weight is two, if I upend this to this the edge weight is sorry edge weight is one for upend this the edge weight is still be one the other computational angle will tell me that the computational involved in adding an edge between this and this is q^3 plus q^2 minus q . If I upend this it is q^3 plus q^2 minus q again. So, it will does in mater so this is I am just going to pick arbiterly and I am going to write $x_2 \times x_5$.

I am going to put that here and I am going to put edge weight of one then I am going to put down here and put down $x_3 \times x_6$ and again edge weight is one and that gives me in just one node left, which is $x_4 \times x_5 \times x_6$ and I upend this I can upend this any one of this the Prim's greedy algorithm, in each case say well since your edge weight is going to be one. No matter of very upend this I do not care the computational complexity will upend this well I also do not care because if I attach this complexity is q^2 plus q^3 minus q .

So, now at this point if I apply Prim's algorithm it does not make difference as I just pointed out how I connect this. So, I am just going to make arbiterly decision, I am going to put down $x_4 \times x_5 \times x_6$ over here. The intersection with this as edge weight one. So, now in this graph if I actually look at so let us call this graph as always g_i , notice that this is particular graph I have that the node weight of the graph will that is sum of these node weight. So, that is 3 5 7 9 and 12. So, this is equal to 12. The edge weight of the graph is equal to four, you can see that this by counting the reds that we have for the difference, the difference is equal to eight which is greater than the number of variables. So, what is that mean?

So, that means that in this particular instance we tried out best to construct a junction tree and we know, if you were able to construct a junction tree then that junction tree would be found by constructing a maximal weight spanning tree and Prim's algorithm was guarantee to give you maximum weight spanning tree and meet in just that. So, we tried out best and we fail. So, what is that means, that means that we cannot construct a junction tree in the ordinary way through Prim's algorithm.

There is in other method however, we may or may not have time to discussion. So, the point here is that, so in this lecture just to summarize quickly. What we trying to do is show examples of how junction trees could be constructed and in one case in sometimes, you can and sometimes you cannot and that is exactly what we illustrated. So, I think that is a good point will stop and will continue in next lecture. Thank you.