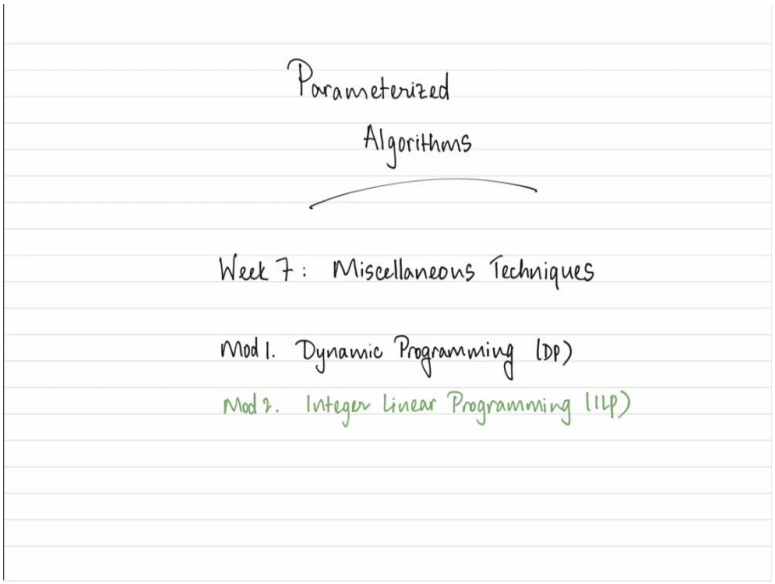
Parameterized Algorithms
Prof. Neeldhara Misra
Prof. Saket Saurabh
Department of Computer Science Engineering
Indian Institute of Technology – Gandhinagar IMSC

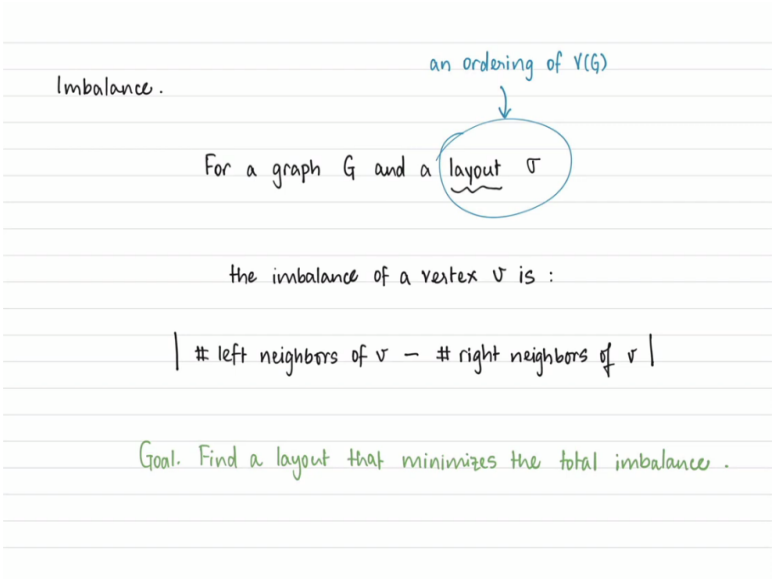
Lecture – 34
ILP for Imbalance Parameterized by Vertex Cover

(Refer Slide Time: 00:11)



Welcome back to the concluding segment of the second module in week 7 of parameterized algorithms. This is going to be our final example demonstrating that ILP can be a really powerful toolkit for coming up with FPT algorithms. So, we are going to switch gears a little bit from computational social choice back to graph algorithms and in particular we are going to be talking about a problem called imbalance.

(Refer Slide Time: 00:33)



So, most graph optimization problems that we work with we are typically trying to find a subset of vertices, there is a subset of agents that can do something for us. In this case we are trying to find a permutation and of the vertex set that optimizes a particular objective. So, this falls under the category of the so called graph layout problems and there are many of these and we are going to work with imbalance as an example.

But it turns out that the techniques that we discuss here also applied quite naturally to some other layout problems as well. So, if that is something that you are curious about than you can look up reference that I have linked to under description for more examples of this same framework playing out in other context for other layout problems as well, but in the meantime let us get back to imbalance and let us begin with the definition here.

Suppose you have an undirected graph G we will assume that the graph is simple and un-weighted and so on and suppose we have a layout sigma so by layout I simply mean a permutation or an ordering of the vertex set of the graph G. So, with respect to a fixed ordering sigma the imbalance of vertex v is the difference between the number of neighbors that it has to its left and the number of neighbors that it has to its right.

So, it is the absolute difference and just to may be a little bit explicit about what I mean by this left right business think about the ordering and think about where this vertex v appears on this ordering. Let us say it appears in the 10th position in this ordering then there are 9 vertices to the left of v that appear before v in this ordering and all the remaining vertices appear after v.

So, I am going to count how many neighbors of v land before it and how many neighbors land after it and absolute difference between these two numbers is the imbalance of v. If this is not clear yet we will look at a specific example in just a moment and hopefully it will become a bit clearer than, but in terms of the optimization goal essentially what we are interested in finding the layout that optimizes the total imbalance.

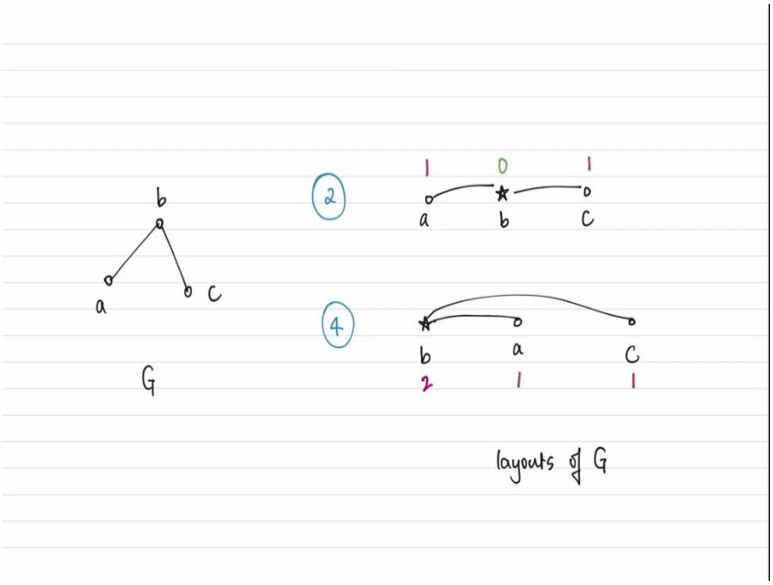
So, the imbalance of a graph with respect to a particular layout is going to be the sum of the imbalances of all the vertices with respect to that layout and we are trying to find the layout that minimizes this total imbalance. So, that is kind of the goal and if you think about a Naïve way to do this of course you could try and go over all permutations of the vertex said that would cost you n factorial.

But once you are given a permutation it is quite straightforward to go ahead and calculate the imbalance of that particular layout. So, that would be the natural Brute Force Algorithms and you could also spend a moment here thinking about how would you parameterize this problem. So, let tell you upfront that as an optimization problem if you consider the decision version which basically ask is that a layout whose imbalance at most K.

This problem is envy complete and you could consider a number of different parameters here. You just (()) (03:31) so you could try to parameterize imbalance (()) (03:34), you could try to parameterize by the standard parameter which is just imbalance parameterized by imbalance and so on and these are all valid parameterizations that have actually been studied in the literature.

But for this particular example when we are trying to resolve this using ILP a more natural parameter to work with is going to be vertex cover. So, we are going to be parameterizing structurally this time and we will be working with the vertex cover number as our choice of parameter for this problem. So, with that said let us take a look at quick example just to make sure that we are on the same page about the definition of imbalance.

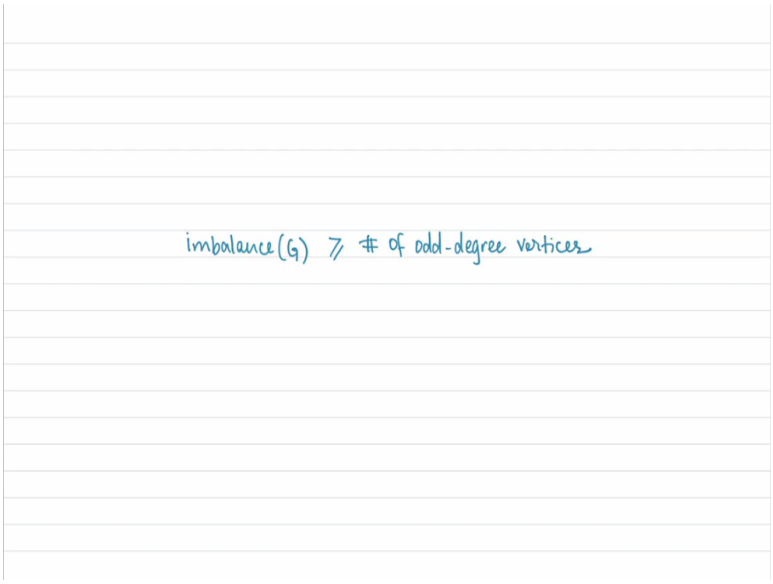
(Refer Slide Time: 04:09)



So, here we have a graph to the left which is just a P 3 it is a path in 3 vertices and there are 6 possible layouts to consider. Let us just look at 2 of them which have contrasting properties in terms of imbalance. The first one we use natural you take the vertex that is in the middle of the path and you put it in the middle so you essentially following the path and you will see that the middle vertex has an imbalance of 0 because it has one vertex to its left and one vertex to its right that happens to be its neighbor.

The corner vertices the degree 1 vertices have an imbalance of 1 each. So, the total imbalance of this layout happens to be 2. On the other hand if you push the center vertex and put it in a corner if you put it at one of the ends of the layout then it is going to suffer an imbalance of 2 and the pendent vertices a and c will anyway have an imbalance of 1. So, the total imbalance of this layout is 4 and so between the two you know which one you want.

(Refer Slide Time: 05:03)

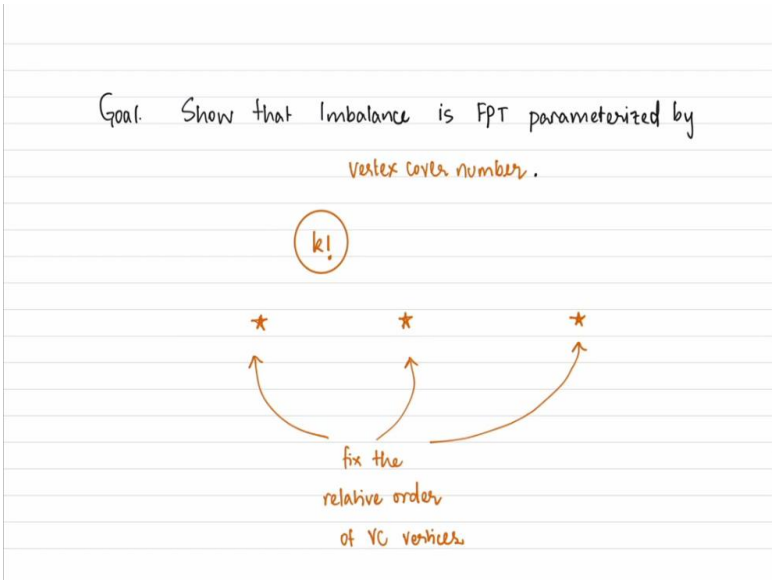


So, in general this might prompt the observation that if you have an odd degree vertex then no matter how cleverly place it is going to incur an imbalance of at least 1 because you will never be able to evenly distribute its vertices its neighbors in any layout. So, the imbalance is definitely lower bounded by the number of odd degree vertices in the graph and in particular in the previous graph we had two degree 1 vertices.

So, we knew that the imbalance will be at least 2 anyway and therefore the layout that we had on the top was actually optimal because the imbalance of this layout is exactly to the one vertex that could have had a 0 imbalances does have 0 imbalance so that is great. You can play around with some examples just to get some intuition for the notion of imbalance what is the imbalance of a path?

What is the imbalance of a cycle? Some favorite trees that you might have as examples or the complete graph and so on and so forth just to get a feel for what the definition is doing. I think that is an useful exercise to do and you should feel free to pause here and just get use to this notion especially if you are seeing it for the first time.

(Refer Slide Time: 06:10)



So, in the meantime let us move on and talk about what we want to do here as I just mentioned we want to come up with an FPT algorithm for imbalance when parameterized by the size of the vertex cover of the graph G. Now you might wonder if the vertex cover is given to us as a part of the input. Well just like with (()) (06:30) for instance you could assume that a 3D composition is available to you because if it was not you can compute one anyway.

It is kind of similar here if a vertex cover has not been given to you can you run your favorite to approximation algorithm or your FPT algorithm to compute a vertex cover and then work with it, you would still be well within the bounce that you are interested in, but on the other hand it is a common assumption with structural parameterizations that the structure that you parameterize and buys actually given to you for free as a part of the input.

So, we are going to assume that we are parameterizing by the size of the vertex cover and the vertex cover of said size has actually been given to us for free. Now the question is how can we make use of this vertex cover to come up an imbalance optimizing layout? Well, the vertex cover is the one thing that we have that is small so let us see if we can play around it. So, to begin with the natural thing to do is to guess the ordering on the cover vertices.

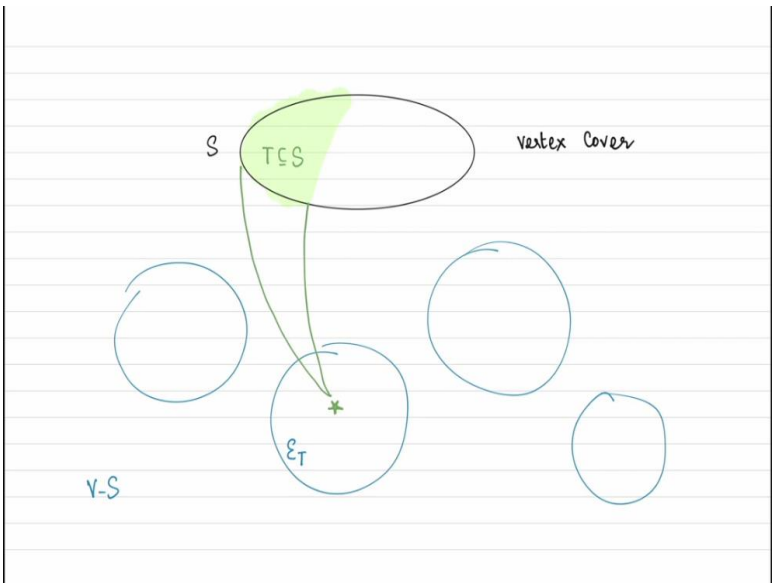
So, this will only cost us K factorial and you could just basically loop over all possible ways in which the cover vertices could be ordered amongst themselves. So, the idea here is that the optimal layout will impose some relative ordering on the vertices of the vertex cover. So, by

going through this exercise we essentially giving ourselves free access to that particular ordering simply because we are trying all of them.

So, one of these guesses is going to be the lucky one. So, I am just going to assume going forward that we have a fixed ordering of the cover vertices and now we are trying to figure out how to sneak in from the vertices from the independent set in a way that is optimal. So, this is a bit that really need some work because the independent set of course could be large and unbounded.

And we do not really have the flexibility to do anything Brute Force like we did with the cover vertices, but just like we have been doing with the previous two problems in the context of ILP formulations we want to see if we can coalesce some of the information coming in from the independent set into manageable groups and remember that what is bounded for us is the size of the vertex cover.

(Refer Slide Time: 08:36)



So, I think it is reasonably a natural thing to try and group the vertices of the independent set based on their interactions with the vertex cover. So, what we are going to do is basically say that okay we have this vertex cover which has K vertices and we are going to look at the independent set vertices and classify them into types based on their neighborhood in the vertex cover.

So, essentially we group together for every subset of the vertex cover. We group together all the vertices in the independent set that have that particular set as their exact neighborhood.

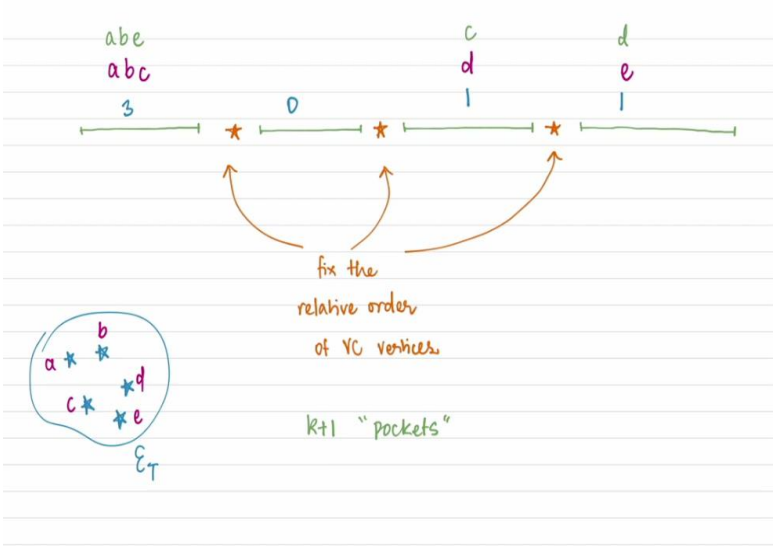
So, let me try saying that again, but this time with some notation. So, fix a subset T of the vertex cover S and now let us go and collect all the vertices in the independent set V – S which are such that their neighborhood is T.

I should have said their neighborhood in the vertex cover is T, but notice that for an independent set vertex the neighborhood in the vertex cover is the same as the neighborhood in the whole graph and independent set vertex cannot have neighbors outside of the vertex cover anyway so it does not really matter which way you say it. So, the idea is that E sub T as a set is going to collect all of those vertices that have the exact same neighborhood in S.

And that neighborhood is given by T. So, again these vertices here have a shared identity and again we have this concept of types. So, I expect that this will become clearer as we go along, but let me already say that these vertices that belong to a common class here in our classification these vertices can really be treated the same as each other. So, their individual identities will not matter so much.

The only thing that will matter is how many of these vertices land up somewhere which ones will not really matter and we will see how that gives us some savings in terms of how many variables we need to express the imbalance of the layouts that we are going for.

(Refer Slide Time: 10:32)



So, with that said let us go back to this picture where we said that we will guess the relative order of the vertex cover vertices. So, once we have done this kind of gives us a bit of scaffolding or a blue print for how our final layout will look out even though we do not know

the specifics yet. So, essentially these K vertex cover vertices determine k + 1 locations of interest if you like.

So, you have the zone that comes before the very first vertex cover vertex then you have the locations that show up between the first and the second vertex cover vertex between the second the third and so on all the way up to the very end where we have the zone that is given by all locations that come after the last vertex in the vertex cover. Now, if you wanted to describe a layout of V of G which is to say you want to give me an ordering of the entire vertex set of the graph G.

And let us say in particular you want to give me an ordering which is consistent with this ordering on the cover vertices then one way that you can describe this ordering is by telling me how many vertices from each equivalence class of the independent set land up in which location. I claim that this is really all the information that I need to reconstruct the layout that you have in mind.

Of course, you might complain by saying the following. So, suppose I have this equivalence class E sub T and let us say as an example that this equivalence class has 5 vertices and suppose you tell me that the first location here the first pocket has three of these vertices the second one has 0 the third and the fourth pockets each have on vertex from E sub T. So, this is the kind of information that I am looking for.

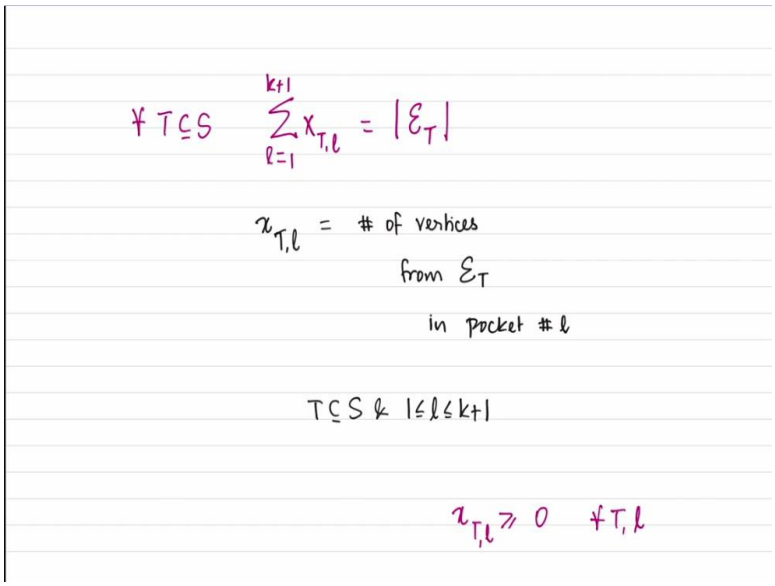
And this is the information based on which I will try to construct a layout, I will try to reconstruct the layout that you have in mind. So, now the point is that what you are telling me is which 3 vertices of E sub T are going into that first pocket. In your layout it could be any 3 vertices and without knowing anything more I might not exactly match your layout on which 3 vertices I select.

But the key observation is that this is not going to matter in the sense that as long as your layout and my layout position the same number of vertices from each equivalence class in each of the pockets the total imbalances of our layouts are going to turn out to being exactly the same. I think this is reasonably intuitive to see because all of these vertices have the exact same neighborhood in the vertex cover.

So no matter which vertices you are place in these pockets they can basically fit in each other shoes quite comfortably. So, if you had a particular layout in mind so just to make this explicit let us give these vertices some names here. So, let us say that these are vertices a, b, c, d and e. So, let us say you placed the vertices a, b, c here and d and e and suppose I did not quite follow this particular choice and instead I had a, b, e here and c here and d here.

Notice that both of these layouts will actually end up having exactly the same imbalances for these 5 vertices because these 5 vertices are seeing the exact same vertices in the vertex cover and they have no other edges to no other vertices at all. So, the independent set vertices do not have edges between them. So, you can freely reorganize these vertices across pockets and even within a pocket if you were to move things around it would not really change anything in terms of imbalance. So, this is really the crucial that dries the whole ILP based formulation.

(Refer Slide Time: 14:25)



This motivates the choice of these variables where we want to know how many vertices from a particular equivalence class land up in a particular pocket. So, for every equivalence class and every choice of location between vertex cover vertices we introduce a variable x sub T, l which tells us how many vertices from the equivalence class E sub T land up at the location l. So, once again notice that T is going to be all possible subsets of the vertex cover and l is going to be all possible locations.

So, notice already that the number of variables that we have introduced is bounded by a function of the vertex cover because the number of types that you can have is at most 2 to k

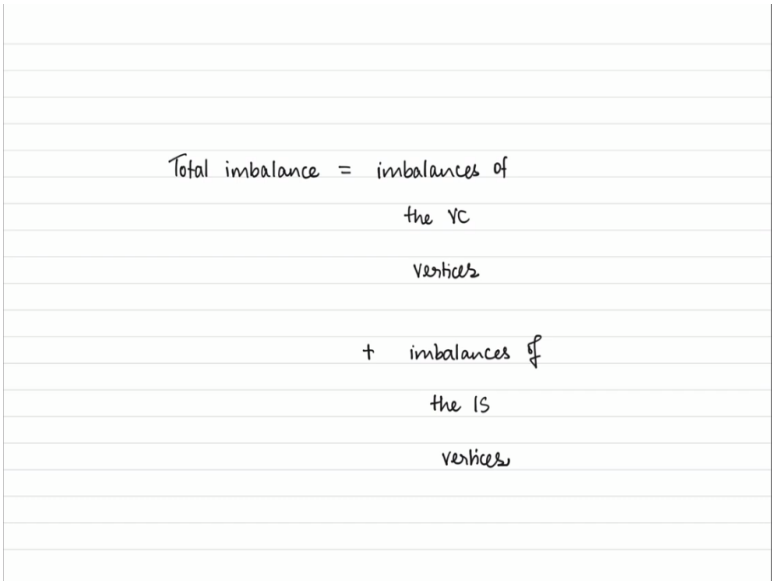
and the number of locations that you have is k + 1. So, the number of variables that we have introduced is 2 to k times k + 1 which is a good start. Now, we do want to introduce some constraints to make sure that these variables lead us to a valid layout.

We are not yet thinking about optimizing for the imbalance. I just want to make sure that these numbers make sense. So, to that end let us throw in these constraints right away. So, we do want to make sure that these values are now negative and on top of that when we fix a particular type and we add up the numbers across locations these numbers must add up to the number of vertices that are there of that type.

It should not fall short if it fall short that means that there is some vertex from that type that is not assigned to any pocket and it should not overshoot these because then you are manifesting vertices that are not even there. So, hopefully this equation this constraint makes sense and that is going to be a first set of constraints just to ensure that these numbers correspond to a valid layout.

So, now with just these constraints we are at a point where any feasible solution to the program that we have come up with so far will actually correspond to a perfectly valid layout. Now we need to encode information about the imbalance of this layout into our program so that the ILP can start optimizing for it.

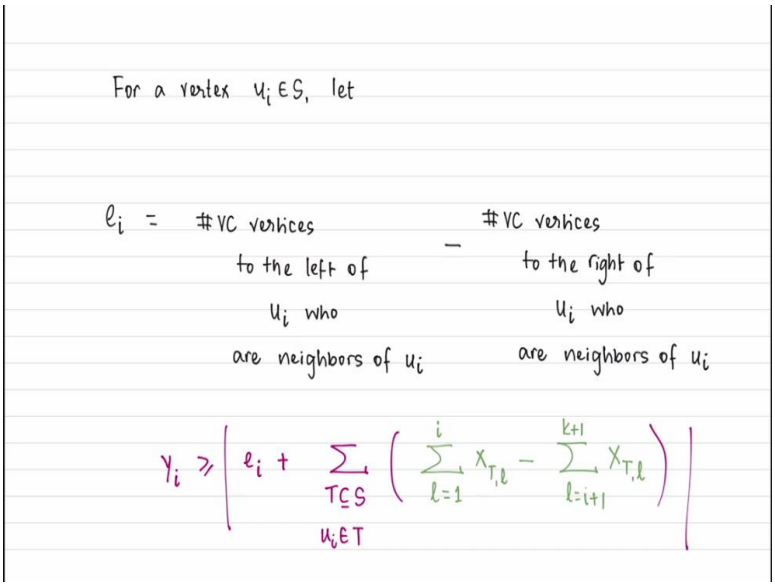
(Refer Slide Time: 16:39)



So, the imbalance of the layout really is the sum of the imbalances of all the vertices and we really have two types of vertices here the vertices that in the vertex cover and the vertices that

are in the independent set. So, let us try and understand how we can compute the imbalances of these two categories of vertices one at a time.

(Refer Slide Time: 17:00)



So, as far as the vertex cover vertices are concerned a lot of their imbalance is already predictable because remember we guessed the relative ordering of the vertex cover vertices. So, we already know how many neighbors of vertices in the vertex cover fall to their left and right as far as the other vertex cover vertices are concerned. So, every vertex in the vertex cover has two kinds of neighbors.

Either neighbors that are also in the vertex cover or neighbors that are in the independent set. Now, about the neighbors that are in the vertex cover we already know their exact locations in terms of who is to the left, who is to the right. So, let us just by looking at the initial guess of the relatively ordering of the vertex cover vertices let us define e i for each vertex cover vertex u i.

So, let us say that u, i was the ith vertex cover vertex in the layout that we have guessed and e i is going to be essentially the imbalance of this vertex coming from its vertex cover friends. This of course does not capture the full imbalance of this vertex because we still have to worry about the neighbors that it has in the independent set. How can we think about the neighbors that it has in the independent sets?

Well let us consider all the subsets of the vertex cover that contain the vertex u i. These are the equivalence classes that we are interested in because for any subset T of the vertex cover

such that u i belongs to T if you look at E T then all the vertices in E T are going to be neighbors of u i because the neighbors of everybody in T and u i belongs to T. So, all the vertices in E T we will want to know where they will land up and will want to make sure that we count the number of vertices from E T that end up in locations behind u i.

And the number of vertices in E T that end up in locations after u i and we take the difference and we have to do this for every equivalence class T that contains the vertex u i. So, that is essentially what this expression is telling us. So, if you just focus on the summation here those outer summation is going over all the equivalence class as T that u i belongs to and the expression in green essentially counts the number of vertices from E sub T that will land up to the left and right of the vertex u i and it takes this difference.

So, e i plus this summation here actually gives us the total imbalance of the vertex u i and you might remember that the imbalances defined as the absolute difference between the left and right neighborhood of a vertex. So, we do need to take this entire expression and put it inside an absolute value expression. So, that is what is being done here. So, those bars here really corresponds to the absolute value notation.

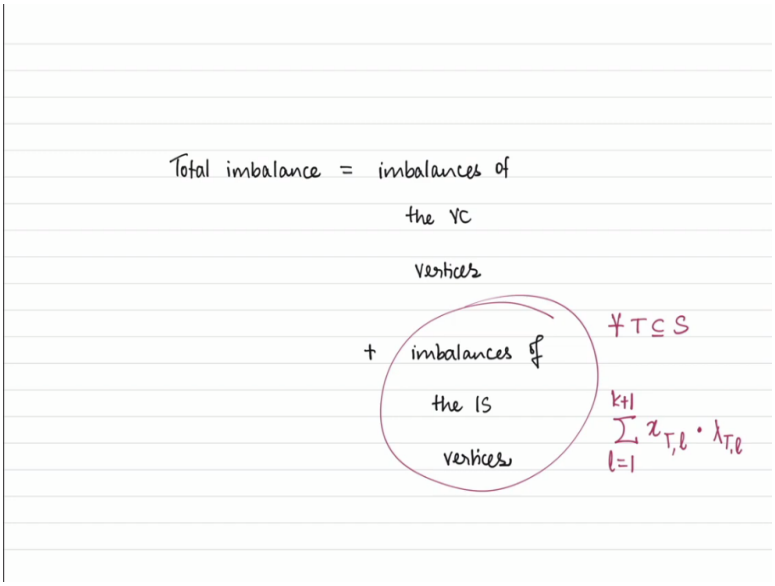
And now what we have also done we have introduced a k variables to track the imbalances of K vertex cover vertices. So, y i is going to capture the imbalance of the vertex u i and so here this constraint is basically saying that y i is at least this. Ultimately, we will throw y i into our optimization objective so that the summation of y i gets minimized and therefore this inequality will really become tight.

Once you have a valid solution that is also optimal y i will end up capturing the exact imbalance of the vertex u i. Another minor point is that you cannot really throw in this absolute value notation into a system of linear equations. So, you will have to rewrite this as two separate equations. In general if you want to express y is at least mod x then you could express that y is at least x and y is at least minus x.

These two inequalities combined will capture the inequalities y is at least mod x so you can write this inequality here without the absolute value sign and then you can copy paste that and have a minus sign go around the expression on the right and those two inequalities

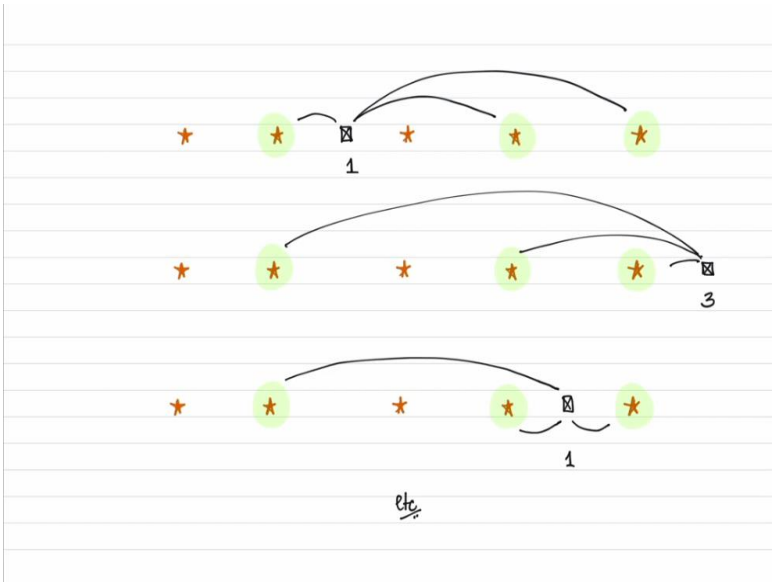
together will capture the one that is shown here. So, with the y i we have kept track of the imbalances of the vertex cover vertices.

(Refer Slide Time: 21:07)



Now let us turn to the imbalances of the independent set vertices. So, to keep track of the imbalances of the independent set vertices it is easiest to think of how the set E sub T get split for every T subset of S. Notice that every independent set vertices belong to one of these E sub T. So, once you have understood this you basically understood the imbalance contribution from all the independent set vertices.

(Refer Slide Time: 21:32)



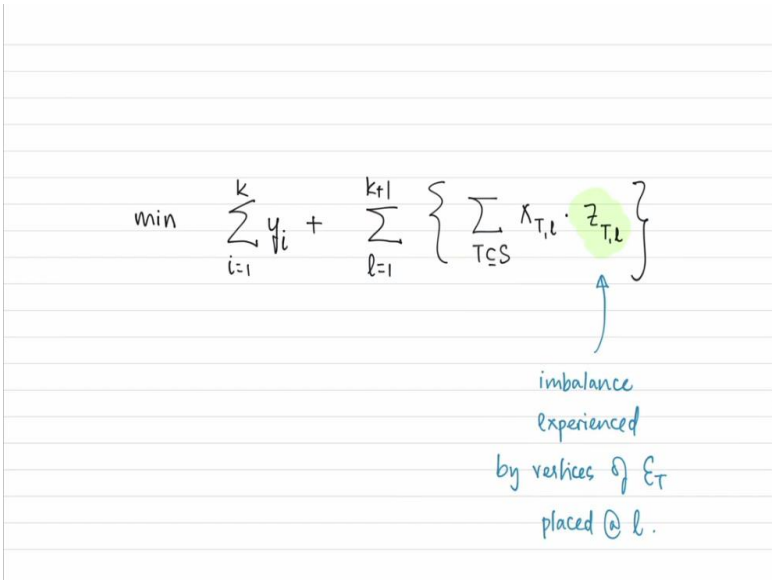
So, let us take a look at this picture here. So, let us say that the orange star vertices represent the vertex cover and the highlighted ones represent sum subset T of the vertex cover. Now notice that if vertex in E sub T lands at any particular location you can fully determine what

its imbalances going to be. It does not matter which vertex you are working with any vertex from E sub T will have exactly the same experience at a particular location.

So, for instance, if you have a vertex from E sub T sitting in the third location as is happening in the first row then it will experience an imbalance of 1 whereas if it was in the last position as is happening in the second row than it experiences an imbalance of 3 and so on. So, this is something that we can pre compute. So, for every T and for every l so for every subset T of the vertex cover S and for every location l from 1 to k + 1 we can figure out what happens if a vertex from the equivalence class E T lands up in location l.

What is the imbalance that it experiences? So, let us go ahead and do that computation and let us call the call the corresponding values that we figure out z sub T, l. So, z sub T, l is the imbalance experienced by a vertex from E sub T when it positioned at location l.

(Refer Slide Time: 22:56)



So, now that we have these constants in place we know that the total imbalance contribution of the independent set vertices can really be thought of as z sub T, l multiplied by the number of vertices from E T that actually land up at location l. So, of course x sub T, l may be 0 in which case that just means that there are no vertices from E sub T at this particular location and then the corresponding z sub T, l will contribute nothing to the sum.

But otherwise you correctly capture the collective imbalance of all the vertices from E sub T that land up at location l. So, only you have to do is sum this overall equivalences classes T and all locations l and you have in aggregate the total imbalance contribution coming in from

the independent set vertices. So, this is it this combined with a y i this expression is actually the totally imbalance of the layout that is dictated by x sub T, l variables.

And this is the quantity that we want to minimize. So, that essentially the entire ILP this is the optimization objective it is subject to 3 constraints 4 if you write them out a little more explicitly, but essentially the first constraint is on the y i basically forcing that imbalance to be at least a certain amount that we know that it should be and then these were the validity constraints to ensure that we actually have a legitimate layout to work with.

So, overall this tells us that the imbalance problem and parameterized by the size of the vertex cover of the input graph is FPT. The actual running time is something that you can get by plugging in the number of variables which is k + 1 times 2 to the k into the expression that we have for the running time of the ILP algorithm at the start of this discussion. So, with that we come to an end of our discussion of ILP based algorithms.

And I hope that it convinced that this is a cool tool to have in your tool kit. Some people like to just think of it as a classification tool of quick and dirty way of just convincing yourself that the problem is at least FPT and then from there on you can try to find it a more combinatorial approach hopefully with a better running time. I think the main deal with why you would not want to stop at the ILP algorithm is because typically I mean especially depending on how many variables you have created the running time can really be quite atrocious.

On the other hand there are a lot of commercial and open source state of the art ILP solves out there that keep getting better every day and they work really well in practice. So, I would say it never hurts to have a valid ILP formulation for whatever problem you are working on if it make sense and if it is natural enough to do because I think these solvers can actually be really useful if your problem needs to be solved in the real world so to speak and at times they may even outperform a combinatorial algorithm which has a theoretically better running time.

So, I would say definitely use ILP as a classification tool for sure, but it can be useful even beyond that. So, again I hope that this is something that you will find useful as you go along and questions and comments as always are welcome either in the comments section on this

video or if you are watching during live run of the course then please do discuss more on either the discord community or the mailing list.

So, I look forward to hearing from you and with this we come to the end of week 7 and we will see you next week with a new theme and a new topic. Thanks for watching and bye for now.