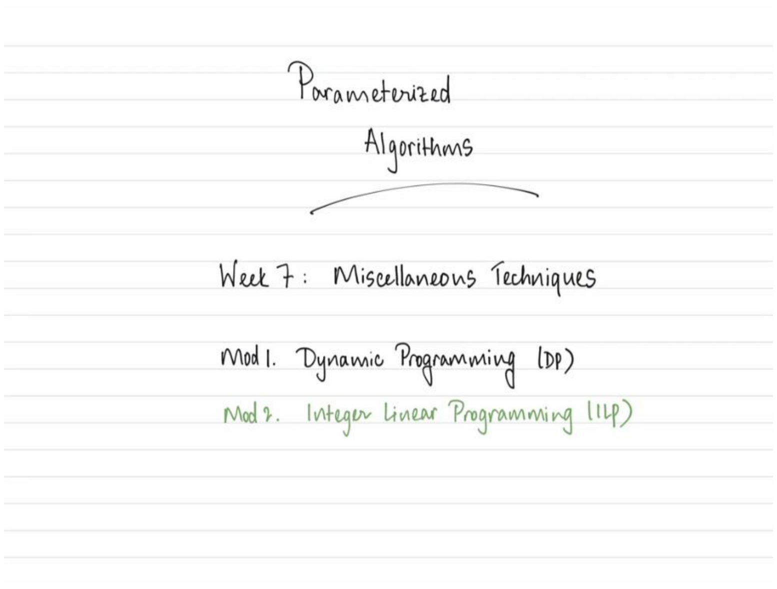
Parameterized Algorithms

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

Lecture – 33
ILP for Envy-Free Allocations and Lobbying

(Refer Slide Time: 00:11)



Parameterized Algorithms

Week 7: Miscellaneous Techniques

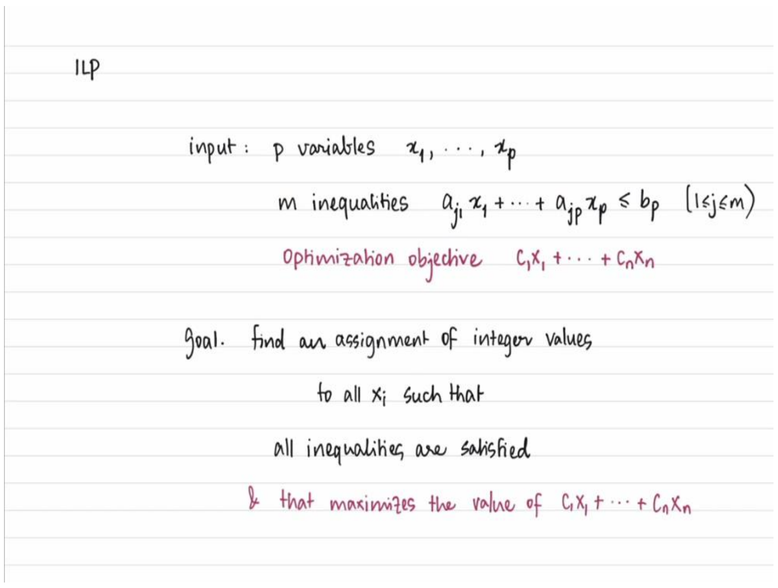
Mod 1. Dynamic Programming (DP)

Mod 2. Integer Linear Programming (ILP)

Welcome back to the second module of the 7th week in parameterized algorithms. This time I want to focus on using integer linear programming as a versatile tool for coming up with FPT algorithms for various problems. You have met ILP conceptually before, you have certainly written LP formulations for problems like vertex cover and used them for (()) (00:33), we have even used LP opt values as an inspiration for an above guarantee parameterization for vertex cover and so on.

Nonetheless to keep this discussion self-contained let us start off with defining what integer linear program is.

(Refer Slide Time: 00:50)



ILP

input: p variables $x_1, \dots, x_p$

m inequalities $a_{j1} x_1 + \cdots + a_{jp} x_p \le b_p$ $(1 \le j \le m)$

Optimization objective $c_1 x_1 + \cdots + c_n x_n$

goal. find an assignment of integer values

to all x_i such that

all inequalities are satisfied

& that maximizes the value of $c_1 x_1 + \cdots + c_n x_n$

Let me begin by talking about the ILP feasibility problem. So, here the input is a collection of p variables and a system of m constraints over these variables described with these inequalities here. Part of the integer in integer linear programming comes from the fact that these coefficients are supposed to be integers the a i’s and the b’s here and apart from that we are also expected to come up with a integer solution to this system which is to say that we want to find them assignment of integer values to each of these variables.

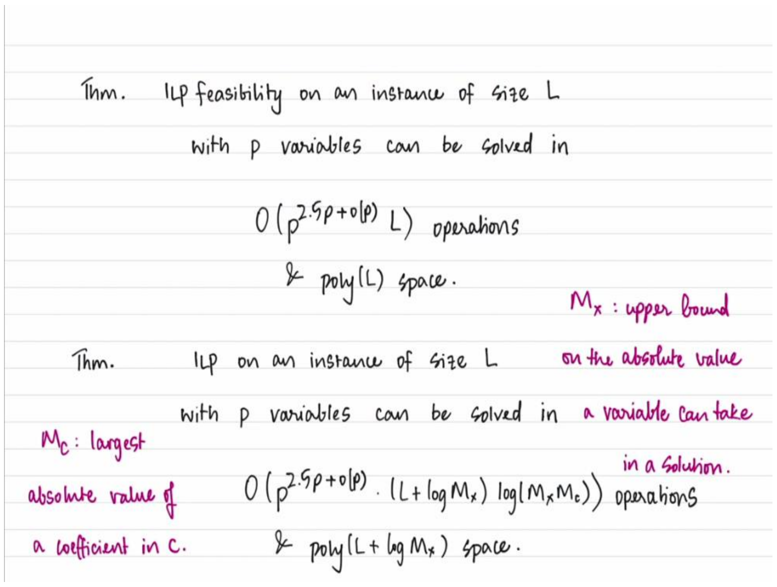
Such that all of these inequalities are satisfied. So, that is the ILP feasibility problem and sometimes you also work with an optimization version of this problem which is simply called integer linear programming. So, just like in the feasibility we are given a connection of p variables and a system of m inequalities over them, but on top of that you also have an objective function which you are trying to optimize for.

So, this objective function again can be thought of as a linear combination of the variables involved again with integer coefficients. So, we just going to think of this as being c times x and the goal like before is to come up with an assignment of integers to the variables so that the system of an equalities is satisfied, but beyond that we also want to optimize the value of the objective function.

Now just to be specific here I have said that we want to maximize this objective, but you could also define a variant when you want to minimize it instead and that would be just as well defined and in fact the algorithms that are known for solving ILP work equally well on both versions. So, you could be in a situation where you want to maximize something or you could be in a situation where you want to minimize something.

In both cases you should be able to invoke ILP as a tool. So, one of the reasons that ILP works in the context of parameterized algorithms is the fact that both ILP feasibility and ILP admit FPT algorithms where the parameter is the number of variables. So, this is the (()) (03:04) result that all of our applications are going to rely on.

(Refer Slide Time: 03:08)



Thm. ILP feasibility on an instance of size L with p variables can be solved in

$O(p^{2.5p+o(p)} L)$ operations & poly(L) space.

M_x : upper bound on the absolute value a variable can take in a solution.

Thm. ILP on an instance of size L with p variables can be solved in

$O(p^{2.5p+o(p)} \cdot (L + \log M_x) \log(M_x M_c))$ operations & poly$(L + \log M_x)$ space.

M_c : largest absolute value of a coefficient in C.

So, in particular ILP feasibility has an algorithm that runs in time some function of p being the number of variables and there is a polynomial overhead in the size of the instance and it is also a polynomial space algorithm. Now, if you had access to the algorithm that solves ILP feasibility for you as a black box then you can use this to also solve the ILP problem by performing a binary search over the range of possible values that your optimization objective can possibly take.

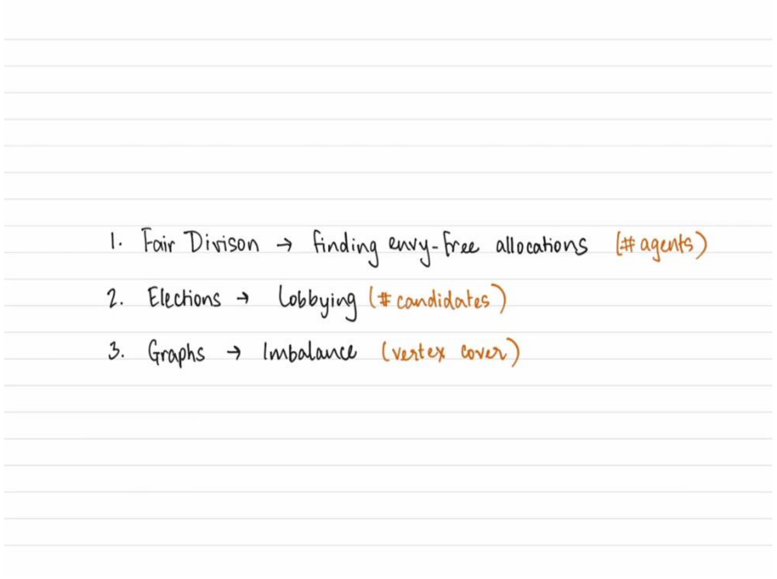
When you guess a particular value for your optimization objective you simply add one more constraint to the system and check for its feasibility and depending on the output of ILP feasibility you can truncate the range one way or the other. So, this is a standard binary search approach and this is what adds a logarithmic overhead on top of the running time that we already have for ILP feasibility and the logarithmic overhead is in the magnitude of the values that your variables can take.

So, I would welcome you to work with the details of how you would use ILP feasibility as a black box to solve the ILP problem and be satisfied with the running time that has been claimed here. Now exploring the mechanics of the ILP feasibility algorithm is beyond the scope of this discussion. So, going forward we are going to assume that we have the algorithm for ILP feasibility as a black box.

And we just going to assume that ILP can therefore be solved in time that is FPT in the number of variables and every time you invoke ILP there are two important Steiner tree checks that you need to do. The first thing is that the number of variables should be bounded by a function of the parameter that you are working with and the second thing is that the absolute values that the variables can take should be bounded in an appropriate way.

So that the polynomial overhead that you have here is indeed a polynomial in the size of your instance. So, with those two things in mind let us go ahead and look at some applications of this framework.

(Refer Slide Time: 05:16)

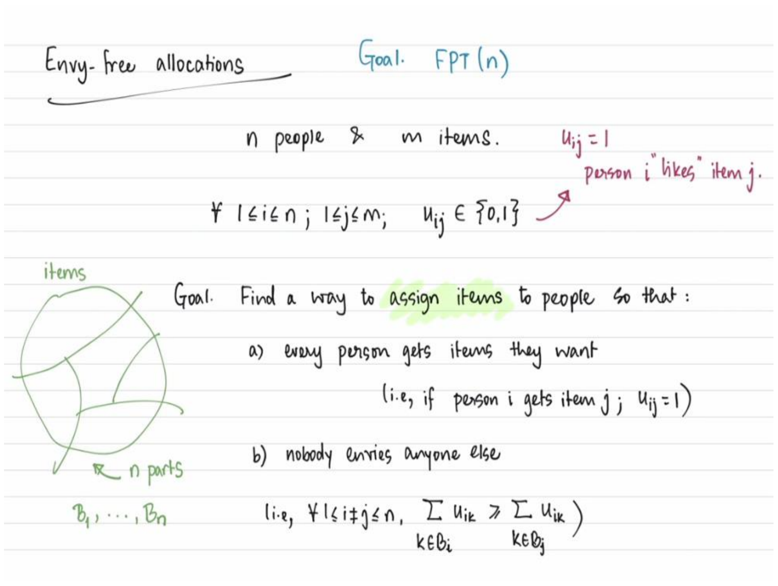


1. Fair Divison → finding envy-free allocations (# agents)
2. Elections → Lobbying (# candidates)
3. Graphs → Imbalance (vertex cover)

We are going to be looking at 3 problems. The first one comes from the area of fair division and this is the problem of finding envy free allocations in a particular setting. The second is inspired by elections and its manipulative problem called lobbying and the third problem is from the familiar territory of graph algorithms and it is the problem called imbalance. So, this module is split into two segments.

In the first one we will talk about finding efficient allocations and lobbying and in the second segment we will talk about imbalance. So, let us get started with the problem of finding envy free allocations. Now this problem has been studied in many, many different settings and we are going to narrow down on very specific scenario that seems to be particularly conductive to this ILP kind of approach.

(Refer Slide Time: 06:05)


items
n parts
$B_1, \dots, B_n$

Envy-free allocations

Goal. FPT (n)

n people & m items.

$u_{ij} = 1$

person i "likes" item j.

$\forall\ 1 \le i \le n;\ 1 \le j \le m;\ u_{ij} \in \{0,1\}$

Goal. Find a way to assign items to people so that:

a) every person gets items they want

(i.e, if person i gets item j ; $u_{ij} = 1$)

b) nobody envies anyone else

(i.e, $\forall 1 \le i \ne j \le n,\ \sum_{k \in B_i} u_{ik} \geqslant \sum_{k \in B_j} u_{ik}$)

So, what we are given as input is a set of n people and a connection of m item and we are also given a utility metric u which describes the utility that each individual has for any of the given items. So, in particular we have these values u i j for I in the range 1 to n and j in the range of 1 to m and u i j is 1 means that person i likes item j and u i j being 0 implies that person i does not care for item j.

That is the semantic interpretation of these Boolean values that you see on your screen here. So, that is the input and what is the goal? The goal is to distribute these items among the people who are here. So, we want to give everybody some subset of items and of course you can do this in very many ways and you can imagine that we have some objectives in mind. So, of course we do want this allocation of items to be a perfect partition of the items which is to say that every time is assigned to exactly one person.

So, in particular nothing is left out nothing is shared, but it is possible that a person goes empty handed, it is possible that a person does not receive any item at all. It is also possible that some person may receive every item that there is so all of these would be feasible allocations, but we do want the assignment to be complete in that no item is left unassigned and we also wanted to be unambiguous which is to say that every item goes to exactly one person there is no confusion.

So, that is what we are looking for, but we are looking for allocation which has some special properties as well. So, first of all we want that every person gets an item that they like. So, they get an item for which they have a utility of 1. This is something that guarantees broader property called Pareto efficiency. So, in case you are interested you could that up, but for now we will just go with what is more I think directly called non wastefulness.

So, this just means that you never gave an item to a person that does not care for it. So, that is the first property this has matching sort of a vibe you could imagine setting up a graph where you have edges based on the utilities and it seems like we are looking for some sort of a generalized matching that gives everybody some collection of items that they like. So, far that is all that we are asking for.

But there is a second constraint that brings the envy freeness that is there in the name of the problem into the picture here. So, we want to make sure that nobody envies anybody else. What does this mean? This means that you consider any person let us say person i and we have person i evaluate the bundles that everybody else got. So, by a bundle I just mean the set of items that were given to other people.

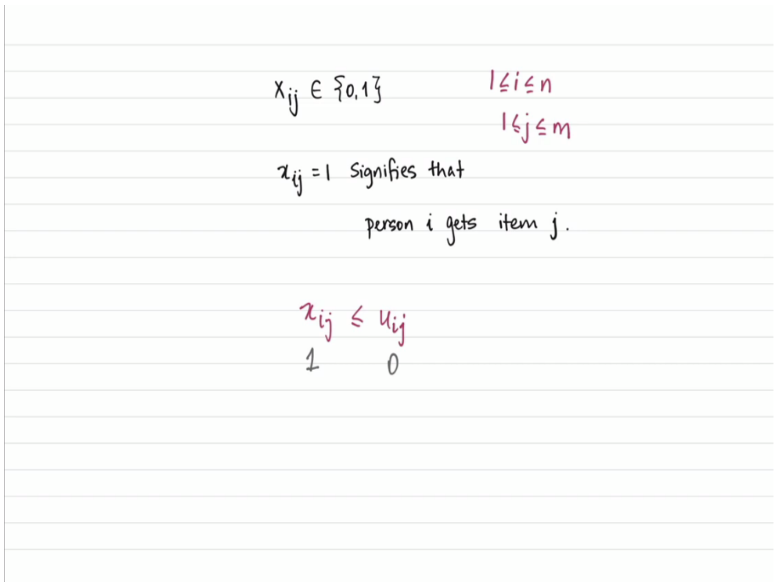
So, when person i is focused on peeking into person j's bundle he is going to see a collection of items and we are going to have him evaluated according to his own utilities not the utilities of person j because she may have a completely different perspective on her bundle compared to person i, but when person i looks at person j's bundle he evaluates it from his perspective and he gets a certain utility for person j's bundle.

And do not envy person j means that the utility that you derive from your own bundle is at least the utility that you see in person j's bundle and this should be true for any other j. So, we want to make sure that no pair of people are in a situation where one person envies the other. By the way it could be that the perspectives are sufficiently different that between two people you could have people who envy each other.

In this setting it would probably make sense for them to exchange bundles and so on, but essentially formally what we are looking for is an allocation which satisfies these two properties on the other hand it is non wasteful gives everybody something that they like and on the other hand we also want to make sure that there is no envy in this system. So, our goal with this problem is to come up with an algorithm that is FPT in the number of people that is n here.

And we are going to try and achieve this by using the ILP toolkit. So, the first thing is to come up with some set of variables that will hopefully capture this problem and will guide you to a solution.

(Refer Slide Time: 10:55)



So, the most natural set of variables seem to be variables that directly correspond to an allocation. So, for instance, we could say that we have variables X i j for i ranging from 1 to n and j ranging from 1 to m and we can say that X i j being 1 signifies that person i gets item j and if it is 0 then person i does not get item j. Now you could write some constraints to ensure that x i j corresponds to valid allocation in the sense that you could easily setup constraints to ensure that every item is assigned to exactly one person.

And so that ensures that in particular there is no sharing and there is no item that is left out and on top of that you could also write constraints that model our objective. So, for instance, you might want to say something like x i j is at most U i j so that will ensure that if x i j has value 1 which is to say that person i does get item j then it is not the case that person i does not value item j because if person i does not value item j then u i j is 0 but x i j is 1 and that would violate this constraint.

So, you could come up with this constraint to ensure that the allocation is non-wasteful as we said before and similarly you could come up with another constraint to ensure that there is no envy in this system and I think you could work through that as a fun exercise at this point. However, having said all that do you think that this ILP formulation would work? In terms of semantics it would work; it would give you the solution that you are looking for.

But can you spot something that is problematic with this particular formulation? Well, if you remember our goal was to come up with an algorithm that is FPT in the number of people and the number of variables that we have introduced here is the product of n and m. So, it depends also on the number of items which presently could be way more than the number of people.

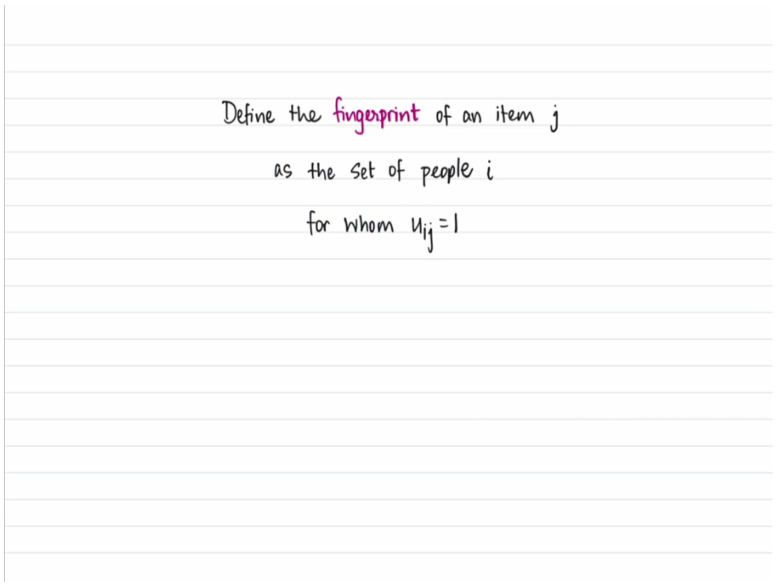
There is nothing that tell us that the number of items is bounded by the number of people in anyway. Therefore, this algorithm while it works does not really have the kind of running time that we are looking for. So, we need a tighter formulation one that hopefully uses a smaller number of variables. So, this would be a good time to take a break and pause and think about how you would either modify this formulation or come up with a different one altogether or maybe even come up with some reduction rules that can help you bound m as a function of n because if you do that then you could apply these reduction rules first.

And then you would have ILP that works out pretty well for you. So, feel free to take a moment here and come back when you are ready. So, hopefully you had a chance to think through this a little bit. Notice that the main bottleneck that we are facing at the moment is the fact that the number of variables depends on the number of items which is apparently non-bounded in anyway by our parameter of interest which is the number of people.

So, it also feels like any ILP formulation that we come up with must involve the items in some way it is not like we can shrunk them off or put them outside the picture somehow. So, a standard idea in this sort of a setting is to see if the set of things it is unbounded can somehow be clubbed into similar categories and maybe the categories can be dealt with at a high level without bothering to basically poke inside.

And hopefully the number of categories is bounded in a way that you want. So, let us make this a little more concrete by defining this notion of a fingerprint.

(Refer Slide Time: 14:46)



Define the fingerprint of an item j

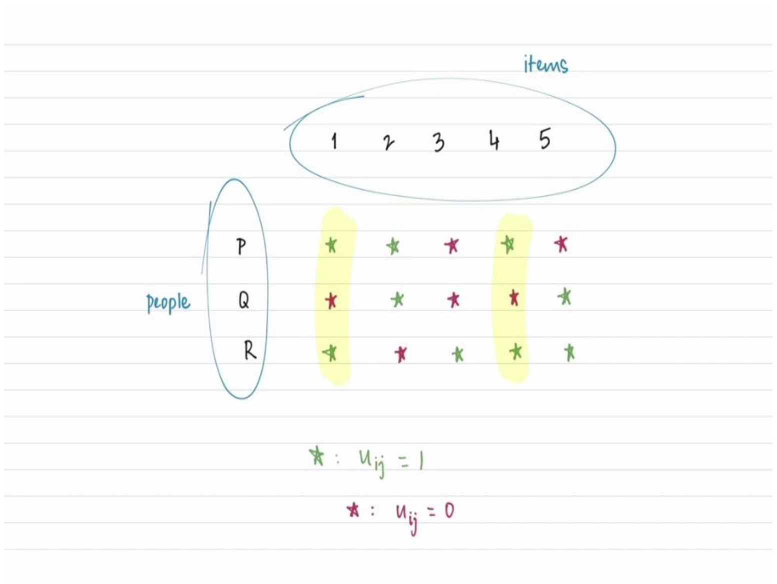
as the set of people i

for whom $u_{ij} = 1$

So, the finger print of an item is the number of people or in fact the set of people who like that item. So, you could have many items that share the same fingerprint, they will have different identities they may go into different bundles and all of that, but in terms of how they are perceived by people they have a shared identity. In particular, if you take a set of items that have the same fingerprint and you go and ask some person about how he or she feels about this collection of items than this person will either reject all of them or accept all of them.

Either this person like all of these items and has the utility of 1 for all of them or he or she dislikes all of these items and has a utility of 0 for all of them that is what having a common fingerprint does and this makes this very useful because now from the perspective of an allocation I really just want to know for each fingerprint I just want to know how many items did a person get from that fingerprint because that is all that I care about if I have the number I can evaluate the utility. Now, just to be sure that we are on the same page about the definition of a fingerprint.

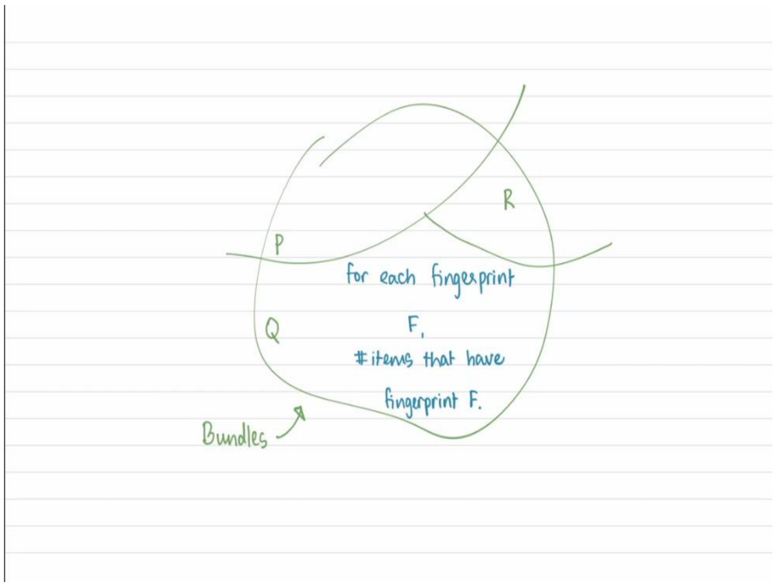
(Refer Slide Time: 16:01)



Let us just look at a quick example. So, let us say you have 5 items labeled 1 through 5 and you have 3 people P, Q, R and let us say that this is the utility matrix with the green stars indicating those situations where the person values the corresponding item and the red stars indicating where the utilities are 0. Now feel free to pause here and see if you can identify whether there are any 2 items here that share a common fingerprint.

So, if you look at this long enough you will probably identify that here are two items that have the same fingerprint in the sense that they are valued by the same set of people namely P and R.

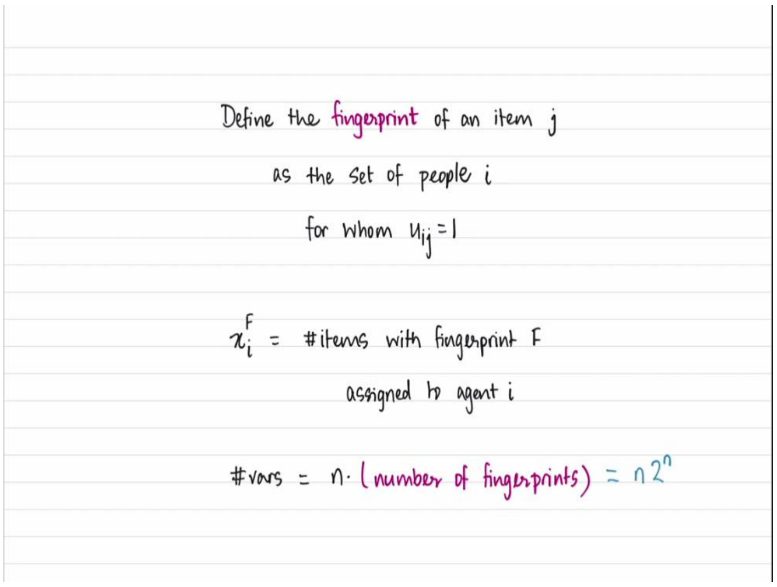
(Refer Slide Time: 16:44)



Now let us just take a quick look at what an allocation would look like in terms of the fingerprint. Now suppose you divided these items between these 3 people P, Q and R and well when I want to analyze Q bundle instead of looking individually through every single item that she may have gotten I basically go through every fingerprint and ask here how many items did you get that have this fingerprint and once I have those numbers that is essentially like a full description of a bundle anyway even though instead of a long list of items it is simply a list that is as long as the number of fingerprints we have.

And you could think of this as being some sort of a (()) (17:22) way of describing the information about these bundles. Now let us see how this translates into the ILP formulation and helps us bound the number of variables.

(Refer Slide Time: 17:32)



Define the fingerprint of an item j

as the set of people i

for whom $u_{ij} = 1$

x_i^F = #items with fingerprint F

assigned to agent i

#vars = $n \cdot$ (number of fingerprints) $= n2^n$

So, in particular let us introduce the variables x sub i F which basically denotes the number of items with fingerprint F that assigned to agent i. This is what we just talked about in terms of understanding bundles and notice that the number of variables this way is going to be n times the number of fingerprints i ranges from 1 to n and F goes over the set of all fingerprints. Also notice that the range of the values that these variables take will now no longer be 0, 1 as we had before.

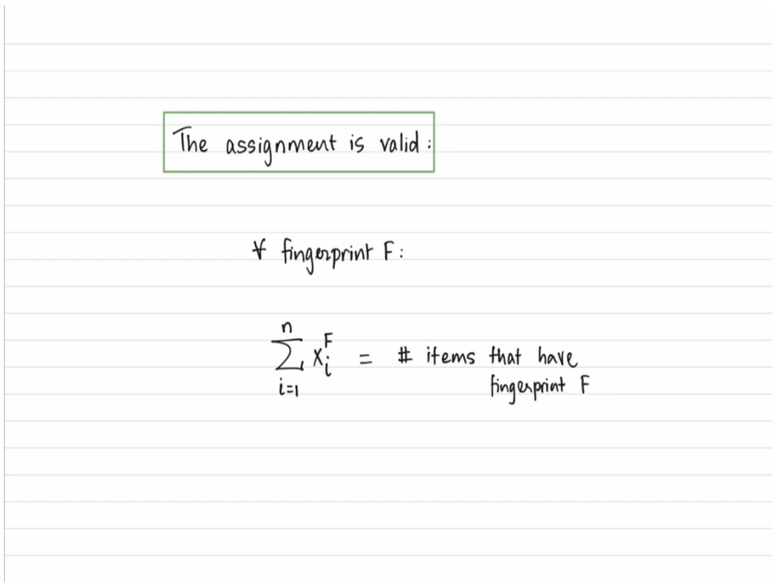
But in fact it will be a value that can be as large as m which is the total number of items. Now, just to be sure that the number of variables now is bounded in a way that we would like. Let us just talk about how many fingerprints there can be. So, we have n people every fingerprint really corresponds to a subset of people and two distinct subsets would correspond to two distinct fingerprints.

So, really the number of fingerprints is equal to the number of subsets that you can form when you have n people. So, that is just going to be 2 to the n. So, the number of variables now becomes n times 2 to the n. Now at this point this may look expensive especially because you know that this is going to get plugged into the running time that we discussed in the beginning of this discussion which had the form p to the order (()) (19:01).

P being the number of variables. So, this really looks like it is going to be a pretty heavy running time, but the point here is that this is a pure function of n as suppose to before when it was not even an exclusive function of n and remember our goal here is to more or less get our first classification result. The focus here is not so much optimizing the running time as it is to see if we can find a nice quick way to identify whether the problem is even FPT in a particular parameter.

So, once we have this bound and if we are able to write out the constraints properly what this tells us is that the problem is FPT in the number of people and that was the goal that we set out for ourselves anyway. So, with that sort of in mind let us go ahead and now try to complete the ILP formulation now that we know that these are the variables that we want to be working with.

(Refer Slide Time: 19:53)


The assignment is valid:

$\forall$ fingerprint F:

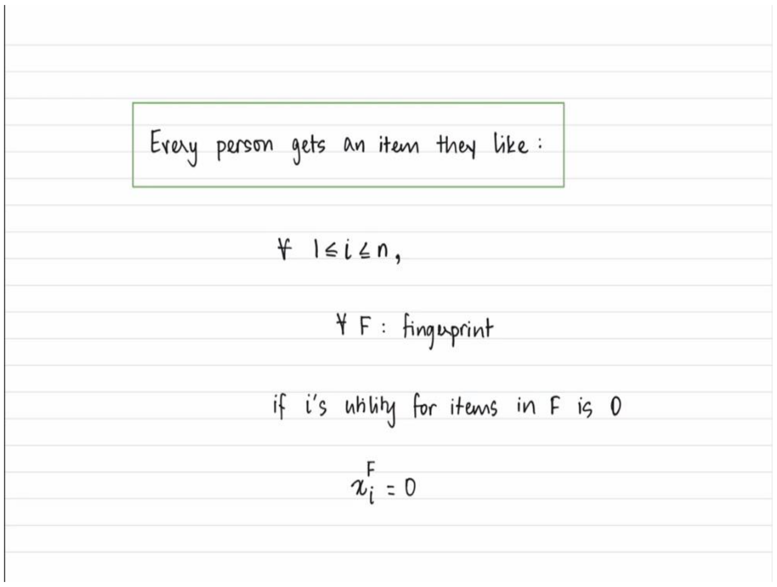
$$\sum_{i=1}^{n} x_i^F = \text{\# items that have fingerprint F}$$

So, the first thing that we want to say is that the assignment is valid. So, these variables take on certain values and we want to make sure that we can actually pull out a valid assignment of items to people from the values that are taken by these variables. Now feel free to pause here and see if you can come up with this constraint yourself and then we can exchange notes once you are back.

So, hopefully you had a chance to try and write this out for yourself here is what this constraint could like. For every fingerprint F we want to make sure that when you sum up the x i F which is to say now we are focused on looking at how all the items that have fingerprint F got distributed among the agents. So, you sum up the x i F over all the agents i and this should add up to the number of items that have a fingerprint of F.

This will ensure that every time that has this fingerprint did get assigned and there was no over assignment there was no sharing. So, this is one constraint that will basically handle ensuring that the assignment that you had is valid. I did say one constraint it is going to be one constraint per fingerprint. So, the total number of constraints that you have introduced here is really actually 2 to the n. What is the next thing that we want?

(Refer Slide Time: 21:13)



Every person gets an item they like :

$\forall\ 1 \leq i \leq n,$

$\forall$ F : fingerprint

if i's utility for items in F is 0

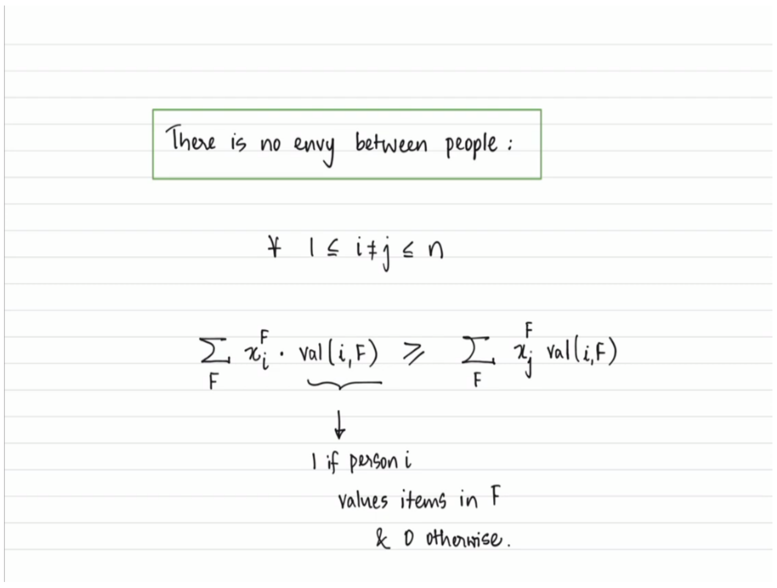
$x_i^F = 0$

We want to say that every person gets an item that they like. Once again this is a cute constraint to figure out. So, feel free to pause here if you would like and then comeback once you had a chance to write it yourself. So, to ensure that every person gets an item that they like let us go every person. So, this is for all i in the range 1 to n and now let us go over all the fingerprint that there are and we asked person i do you all the items that have fingerprint F or not?

Remember the answer is going to be uniform it is never going to be ambiguous. So, every person i by definition either likes every item that has a particular fingerprint or like none of the items that have that fingerprint. This is by the very definition of a fingerprint and we had discussed this a few minutes ago. So, if a person i does not like any of the items that have fingerprint F then for every such combination of i and F we want to introduce the constraint x, i, F equals 0.

So, we just over all of these combination i, F and we introduce this one simple constraint for them. This will just ensure that nobody is assigned an item that they do not like.

(Refer Slide Time: 22:33)



There is no envy between people :

$$\forall \; 1 \le i \ne j \le n$$

$$\sum_F x_i^F \cdot \underbrace{\mathrm{val}(i,F)}_{\substack{\downarrow \\ 1 \text{ if person } i \\ \text{values items in } F \\ \& \; 0 \text{ otherwise.}}} \geqslant \sum_F x_j^F \, \mathrm{val}(i,F)$$

So, the last thing that we want to ensure is that there is no envy in the system. So, when person i looks at j’s bundle and he evaluate it then they find that the utility that they would have derived from that bundle is not more than the utility that they are presently getting from their own bundle as specified by the allocation that we associate with the variables x, i, F. So, once again feel free to try and come up with this constraint yourself.

And then come back to exchange notes once you are done. So, here is one way of writing this constraint. So, we go over every pair of distinct people and this constraint is basically trying to ensure that i does not value j’s bundle. So, we go over essentially again all fingerprint both sides of this inequality we essentially trying to evaluate the value that i has for particular bundle.

So, essentially we want to look at how many items? So, let us look at how a person i is going to evaluate person j's bundle that is the expression of the right hand side. So, first of all we need to figure out how many items did person j get that have fingerprint F. So, we have collected those numbers and essentially the utility that i has for this bundle is more or less going to be the sum of these numbers.

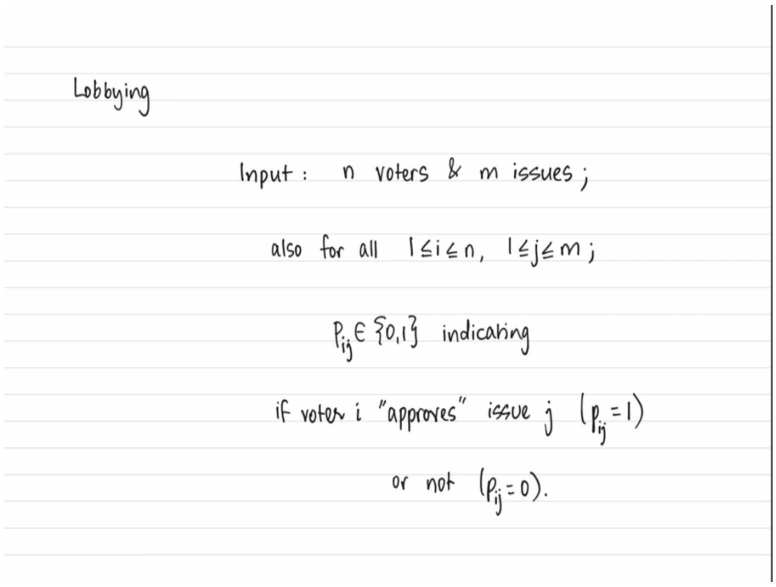
Except we need to ignore the ones that i has no value for. So, that is why we have this multiplier which I am calling i, F which basically is one if i values all items that have fingerprint F and a 0 otherwise. So, the effect of adding this multiplier is that all the irrelevant numbers are just going to get knocked off the summation. In particular, on the right hand side you are only going to count those x i F's corresponding to fingerprints F that i actually cares about.

If i does not care for a fingerprint it does not matter that they belong to j's bundle. So, only the once that matter should count towards the sum and that is exactly what this multiplier is helping you do and it works in the exact same way in the expression on the left hand side of this inequality as well. So, this inequality essentially captures the idea that i does not envy F and if we setup these inequalities for every pair of agents or people i and j.

Then we are pretty much done. So that is the entire ILP formulation. So, we have a whole bunch of constraints over n times to the 2 to the n variables and that gives us the FTP algorithm that we wanted this is FTP in the number of people for the problem of finding envy free non-wasteful allocations in the setting where people have Boolean 0, 1 utilities for a bunch of items.

So, with this we wrap up our discussion on the problem of finding non-wasteful envy free allocations and I would like to move on to the next problem which is lobbying.

(Refer Slide Time: 25:26)


Lobbying

Input: n voters & m issues;

also for all $1 \leq i \leq n$, $1 \leq j \leq m$;

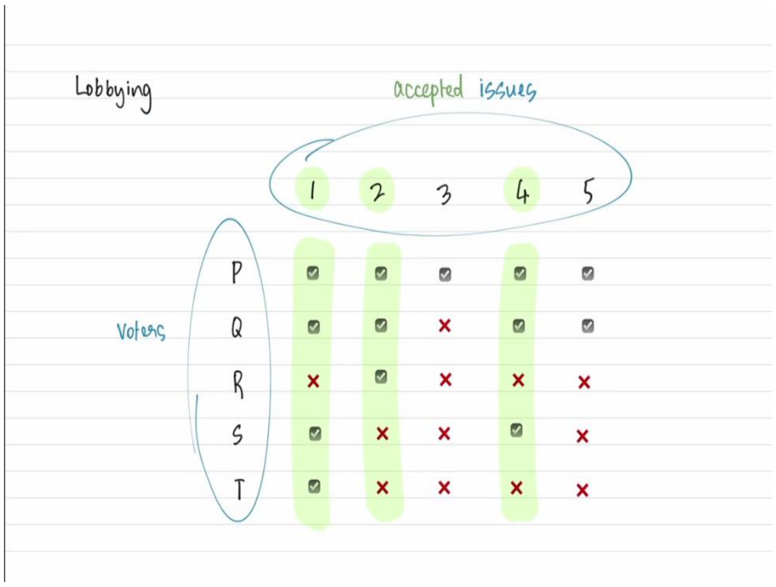
$P_{ij} \in \{0,1\}$ indicating

if voter i "approves" issue j ($p_{ij} = 1$)

or not ($p_{ij} = 0$).

So, in lobbying the setting is the following. We have n voters and m issues and these voters have opinions over these issues and again these opinions are expressed in this binary form. So, just like we had the utility matrix before here we have a preference matrix and P i j is one if voter i approves of issue j and it is 0 otherwise. So, I think this kind of input is helpful to visualize with the help of an example.

(Refer Slide Time: 25:55)


Lobbying

accepted issues

Voters

	1	2	3	4	5
P	✓	✓	✓	✓	✓
Q	✓	✓	✗	✓	✓
R	✗	✓	✗	✗	✗
S	✓	✗	✗	✓	✗
T	✓	✗	✗	✗	✗

So let us take a look at this situation here where we have 5 issues and 5 voters and the entries that correspond to the check marks tell you that the voters are happy with these issues and the crosses tell you that the voters are unhappy or disapproving of the corresponding issues. Now this notion of an issue is of course an abstraction for various applications that you can imagine.

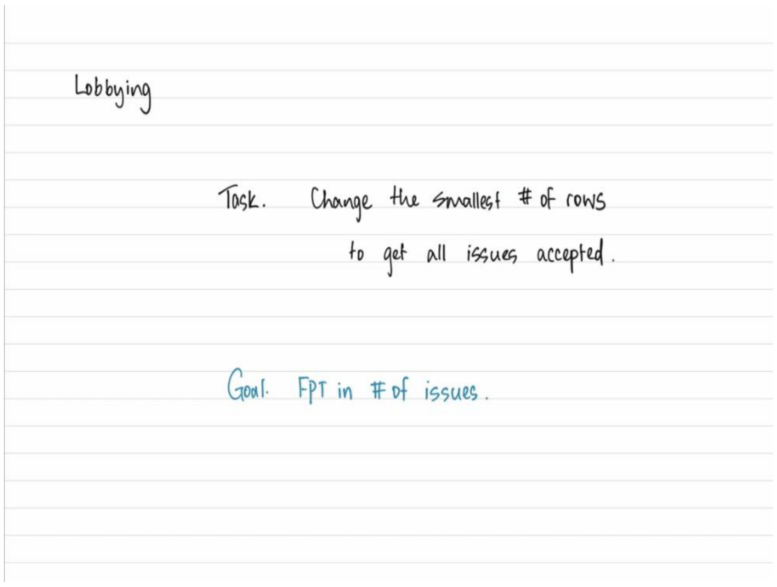
These voters here could be a jury and they are trying to figure out maybe which contestants should go to the next level. It is possible that these voters here are trying to figure out which project should get funded or maybe these issues corresponds to actual problems and we are trying to figure out which one we should prioritize and things like that. So, this is something that has a number of applications.

We are going to go with this assumption that if an issue has a majority vote which is to say that more people vote in favor of the issue than those who do not then these are the issues that are going to get shortlisted or accepted or they are going to move to the next round whatever their setting is. So, we want to let us call these the issues that get accepted. Now what is the lobbying problem?

In the lobbying problem there is an external sort of a force or an agent who wants to come along and make sure that every issue gets accepted. You could come up with a more nuanced version of this problem where you say that somebody wants a specific set of issues to get accepted and a specific set of issues to get rejected. So, this is essentially somebody who wants to manipulate the system and get a particular desired outcome.

So, for simplicity I am going to say that let us say the desired outcome is that we want every issue to get accepted.

(Refer Slide Time: 27:39)



Lobbying

Task. Change the smallest # of rows to get all issues accepted.

Goal. FPT in # of issues.

So, in the lobbying problem what we want to do is see if we can influence a small number of voters so that we can actually achieve our end goal. We are going to assume that once you are able to influence a voter you can ask them to change their vote in any way you like and it should make sense that if you want to get every issue accepted when you have access to a particular voter you just want to change all of their crosses to check mark because that is the natural greedy thing to do and you could never go wrong with that.

So, that is kind of the goal we want to know what is the smallest number of voters that need to be influenced, that can be influenced so that you get the outcome you want. So, again for this simplified problem description I am going to assume that you have equal access to all the voters. So, in more sophisticated models you might say that every voter has a price that you need to pay if you were to try and change their minds.

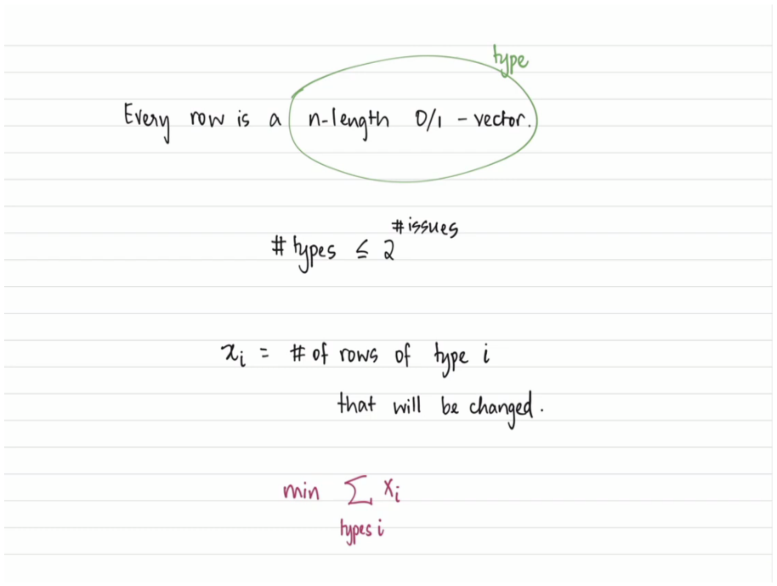
And you are working with the total budget and you need to find a solution that is feasible with respect to such a budget and remember that some voters may just have a price that is not well defined in the sense that there may be voters who will never be influenced. So, a model which involves prices will be able to capture that by setting the prices of such voters to be anything that exceeds the budget or maybe infinity just to demonstrate the principle that there are voters who cannot be influenced.

Now we are not going to be dealing with these variations here which is going to assume that if we were to approach a voter then they are going to be nice and they will be happy to change their opinions for us and this is just because we want a toy problem to demonstrate the ILP technique, but if this sort of problem is of interest then you should definitely look at the vast amount of literature that is there around more interesting ways of modeling realistic scenarios..

So, coming back to our simplified setting our goal will be to come up with an FPT algorithm which is FPT in the number of issues for the lobbying problem. So, in sprit this is not going to be too different from how we approach the previous problem, So, if you would like to take a pause and see if you can come up with your own ILP formulation for this problem this would be a good time to do it.

So, just like we did before remember we came up with this notion of a fingerprint here I want to anticipate that we will need something similar so that we can compress the voter sets. Remember we do not want the number of voters to directly play into the number of variables because that will be too expensive just like before. So, we want to somehow compress that information.

(Refer Slide Time: 30:18)



Every row is a n-length 0/1 - vector.

type

$\# \text{types} \leq 2^{\# \text{issues}}$

x_i = # of rows of type i that will be changed.

$\min \sum_{\text{types } i} x_i$

So it is natural to try and see if we can think about voted types instead of voters. So, what is a type? A type can be thought of as a specific subset of issues and it is one particular opinion and all the voters who share that same opinion they will be of the same type. So, from the point of view of the lobbyist it does not matter which voters be influenced if you are given a set of voters that have the same type.

What I mean is that the specific identities of these voters is a material. If you have 20 voters who have the same opinion if you are able to influence 5 of them it does not matter which 5 of them you influence it will have the same effect. So, once again we are in a situation where we only need to look at all voters who have the same type just in terms of numbers instead of actually going into the set and worrying about specific identities.

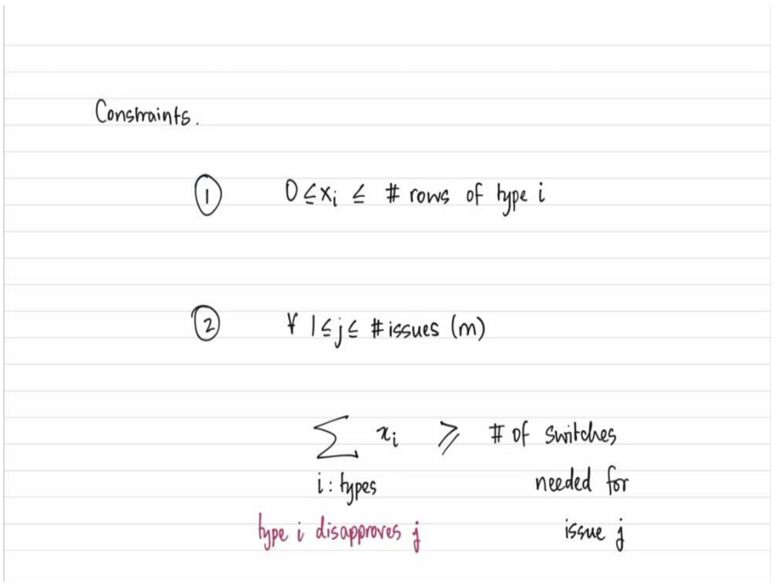
Hopefully, this will become clearer as we get into the actual LP formulation, but for now hopefully the intuition is making sense already. Notice that the number of types is going to be just due to the number of issues because as we said every type is simply a subset of issues and this is the number of subsets that there are and in fact I should say that the number of types and principle is exactly equal to due to the number of issues, but the number of types that are actually realized may be less.

So, it is possible that there is a particular subset of issues for which there is no voter at all who has that particular opinion who approves exactly that subset of issues. So, in that case the number of realizable types, the number of types that actually occur in your instance may even be smaller than this bound that we have. So, now let us see if we can leverage this notion of types to come up with a small set of variables whereby small I simply mean a number that is just a function of the number of issues.

This will also capture this intuition of why identities within a type do not really matter. So, we are going to introduce the variable x i to say that this is the number of rows of type i that will be changed. So, by rows I am referring to people here, but yes x i tells us how many people of type i do you need to influence and what we want to minimize of course is going to be the sum of all of these x i because that is the total number of people that we will end up influencing.

Now we want the values that are taken by these variables to meaningfully correspond to a lobbying solution. So, this is where the constraints come in.

(Refer Slide Time: 32:55)



Constraints.

① $0 \le x_i \le$ # rows of type i

② $\forall\ 1 \le j \le$ #issues (m)

$$\sum_{\substack{i:\text{ types} \\ \text{type } i \text{ disapproves } j}} x_i \geq \text{\# of switches needed for issue } j$$

Our first constraint basically says that you should not try and influence more people than there are actually available to you. So, if (()) (33:03) is the number of people that have type i in the system then x i should be at most (()) (33:10). So, this is to say that the values that are taken by these variables actually correspond to a feasible solution, but notice that this constraint by itself is not enough in the sense that this constraint is not really carry the semantics of the act of lobbying.

We have done nothing to ensure that the issues that are currently not accepted actually get accepted if we were to follow the path laid out by the x i's in some feasible solution to the ILP. In particular with just the constraints that we have currently if you simply said every x i to 0 then you would have satisfied this set of constraints and you would also optimized the objective function, but that is not really a meaningful thing to work with at all.

So, actually encode the act of lobbying we are going to need another set of constraints. So, essentially let us go and look at the issues now. We are concerned about the issues that are not accepted to begin with. The ones that are accepted we are not worried about them none of the changes we make will affect the fact that they were already accepted they will remain accepted any ways.

But if an issue is not accepted to begin with then that is because it does not have enough people vouching for it, enough people approving it. So, let us say that for every issue j you count how much more you need to get the issue accepted. So, let us say you have a pool of 5 people and their sum issue that is accepted only be one voter that means for this issue you need to have at least 2 more people who are currently disapproving to switch their votes and actually approve the issue.

So, let us go through the preference matrix given to us in the input and for every issue identify the number of extra ones that we need for that issue to be accepted and then we are ready to bring in the next constraint. So, for every issue what we want is that the number of changes that we made so this is going over all the types and for an issue j we only go over the types of voters who disapprove of the issue because the ones who already approve of this issue there is no point even if we are going to influence them for this issue they do not really contribute anything they already approved of it anyway.

So, let us go over all of those types which do not approve of this issue to begin with and if we sum up the x i's corresponding to all of these types then this sum should be at least the number of ones that you need for this issue whatever number you identified by looking up the input. So, for the issues that are already accepted there is quantity on the right hand side is going to be 0 and this inequality will be trivially satisfied.

But for the ones that are not accepted to begin with it is these inequalities that will ensure that a feasible solution to this ILP actually corresponds to a lobbying plan that actually works. So, this completes the description of this ILP and notice that we are working with 2 to the n variables here and that is what makes this I am sorry I should have said 2 to the m, m being the number of issues.

And that is what makes the lobbying problem FPT in the number of issues. Now the lobbying problem is fairly well studied as I said even with more sophisticated models and you can imagine that there are many other parameters here in this picture that you could work with. So, I leave you with a reference in the description of this video so that you can look up more about this if you are interested.

I should point out that both of the problems that we discussed here belong to the broad domain of computational social choice and if the flavor of these problem was something that you found interesting then you might want to look up more on computational social choice as a fear and once again there are pointers and links the description of this video that should help you get started with that exploration as well.

Now, we have one brief segment left for this week where we talk about the problem of imbalance parameterized by a vertex cover that will be our last illustration of ILP as a technique for coming up with FPT algorithms and we are going to check that out in this second segment of this module. So, I will see you there.