Artificial Intelligence: Search Methods for Problem Solving

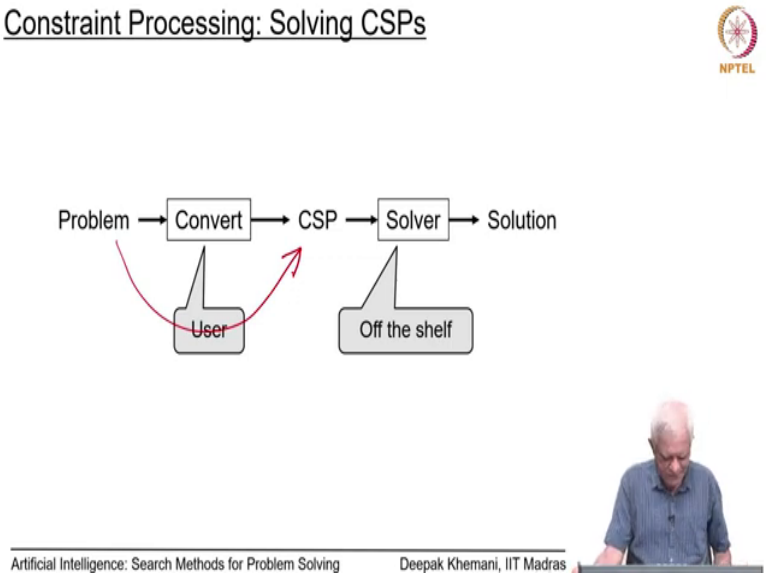
Prof. Deepak Khemani
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Chapter – 09
A First Course in Artificial Intelligence
Lecture – 92
Contrast Processing
Solving CSPs

So, we have been looking at Constraint Satisfaction Problems. And, I hope you are convinced that a whole lot of problems can be expressed as constraint satisfaction problems. And, we saw some examples. Of course, we are confining ourselves to finite domain constraints and we have trying to look at a general way of solving constraints.

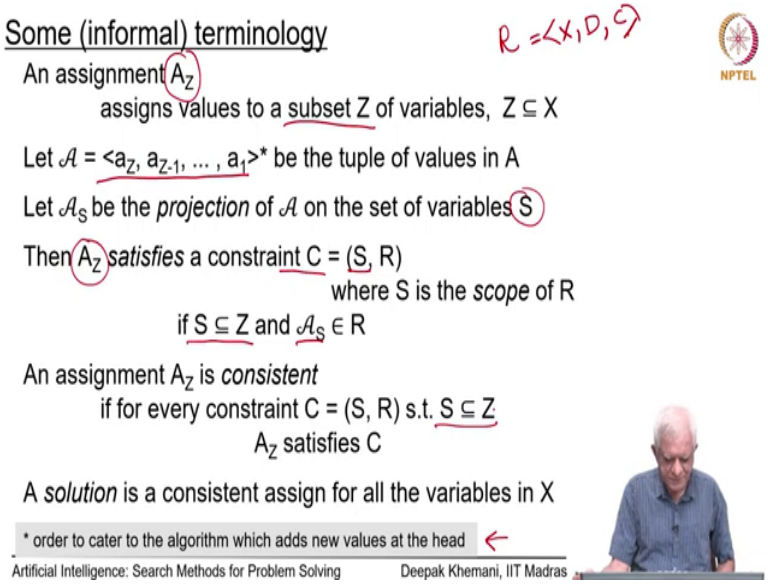
Now, if you have studied things like linear programming or integer programming, you can see that they are also special cases of constraints, but now they also have specialized methods of solving them. But here what we are interested in is general purpose methods of solving constraints.

(Refer Slide Time: 00:58)



So, let us look at that, now onwards. As we said at the close of the last session that, the idea is that we have a problem to solve and we express it as this CSP. And, that is done by a user essentially and then we take a solver and get a solution. And, now we are interested in how to implement such solvers?

(Refer Slide Time: 01:24)



So, let us begin with some informal terminology, which we will use to talk about our algorithms. Now, remember that the task of constraint satisfaction problem is so, what is the constraint satisfaction problem you have a network, which is made up of X and a set of domains, a set of variables, a set of domains and a set of constraints C.

And, for every variable in X there is a corresponding domain in D and the constraints are over some subset of the variables essentially. So, the task is to find an assignment of a value to every variable, such that all the constraints are satisfied. So, we will look at the assignment as a vector in a short form and call it A. In this particular case a subset Z, because we are talking of a subset Z of variables of the X.

So, this assignment is a partial assignment, it is not a complete assignment in the sense a complete assignment would be over all the variables essentially. So, instead of writing 0

equal to A 1 and X 2 is equal to A 2 and so on. We will adopt as is commonly done a simplified form in which we only write the values.

Uh. And, since we are looking at uh set of Z variables, we have Z values in this vector. And, as noted below here, we have written them in the reverse order simply because we will be adding them the new values at the head of a list; we will be treating it as a list essentially.

Otherwise, it has no significance you could have written a 1 to a z, but this is just consistent with the algorithm that we will look at. Now, out of this Z variables, if you have a subset S and we say that the projection of those Z variables of the assignment vector on S is essentially those values which correspond to the set S.

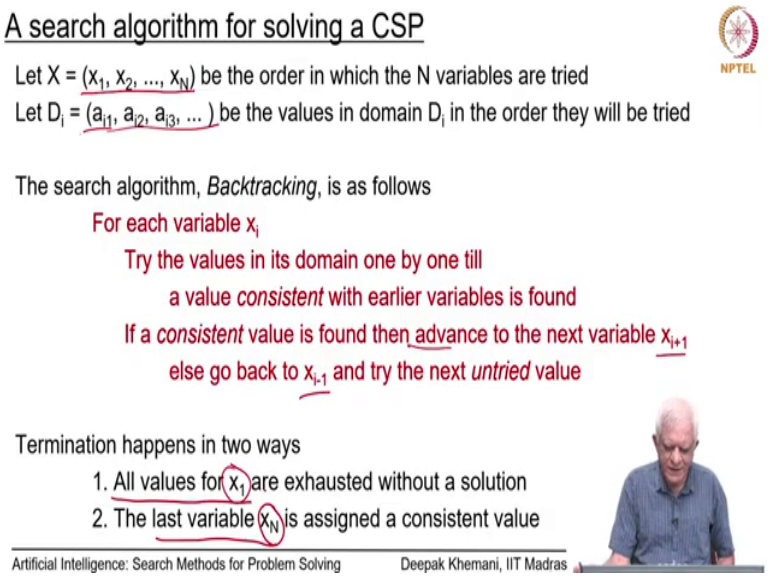
What is this set S? The set S that we are interested in is this set S, where we have a constraint who which has a scope S and the relation R. So, the scope tells you which variables the relation is on, we say that this assignment A Z satisfies a constraint C. If, we take the projection of the assignment on the variables S and they belong to the relation R that were talking about essentially.

So, for the other condition is that, the assignment must cover that scope of the constraint or in other words the scope of the constraint must be a subset of Z essentially. So, that tells us what you mean by satisfies?

So, when assignments satisfies the constraint if in the relation for that constraint we have that projection of the assignment. Then, we say that an assignment is consistent if for every constraint C such that the scope falls within Z a Z satisfies that C essentially hm.

So, essentially we will be interested in consistent assignments in our simple algorithms that we are going to look at. We will go variable by variable, try out a value for a variable and see, if it is consistent with the assignment which has been done so far essentially. And, if we can reach the end then we have a solution, a solution is a consistent assignment for all the variables in X.

(Refer Slide Time: 05:26)



So, let us look at this process. So, as we have been saying our focus in this course has been on search methods. And, we are trying to see that search methods are applicable inside constraints also and constraints in fact, kind of combine search and reasoning. So, we are searching for these variables N variables and let we let us say that we have pre decided the order, that X 1 is first and X 2 is second and so on and so forth.

And, within each domain let us say that we have pre decided the order in which we will look at the values essentially. There are algorithms which do not pre decide this, but which do it on the fly dynamically, but we do not have time to get into those.

So, the search algorithm the basic search algorithm assumes a fixed order and it is called backtracking it simply works like this, that for each variable X i and you will vary i from 1 to

N, try the values in it is domain 1 by 1 till a consistent value is found what do we mean by consistent?

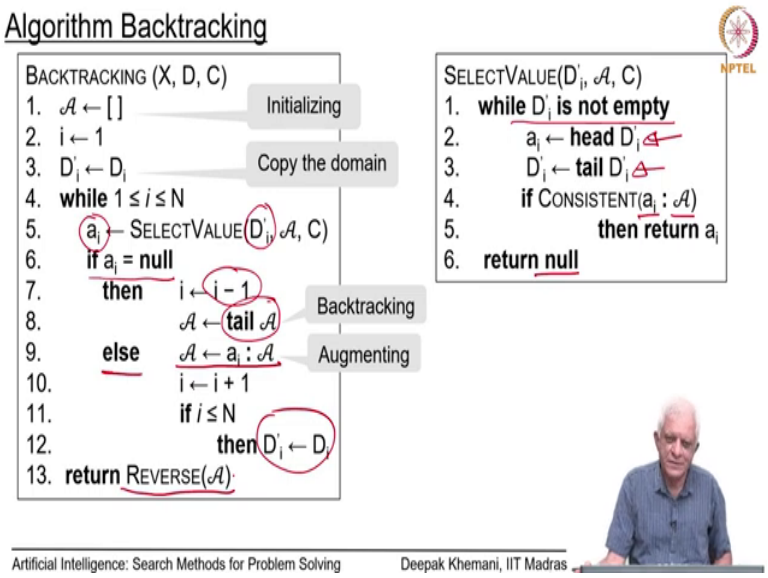
So, that the value for the variable, let us call it X i and all the previous variables together they are consistent which means, they satisfy all the constraints that whose scope falls within this X i variables.

If a consistent value is found, then advance to the next variable X i plus 1, if it is not found then go back to X i minus 1 and try the next untried value. That is the basic idea of the backtracking algorithm and we will look at that in a little bit more detail.

Now, our algorithm will terminate in two ways. One is when it fails and we will know that it has failed, if all values for the first variable are exhausted without finding a solution. What is a solution? A solution is the consistent assignment for all the variables.

So, if we keep backtracking and if we keep backtracking and eventually exhaust all the values in the first variable X 1; that means, there is nothing else left to try and we can report failure. The other condition is a success condition and that happens, when the last variable which is X of N is assigned a consistent value, then we have a solution.

(Refer Slide Time: 07:50)



So, that is the simple backtracking algorithm essentially. So, let us look at this in a little bit more formal way. So, the input to this is a network or a constraint satisfaction problem. We start off by initializing the assignment vector to an empty list. As, we said we will be treating it as a list and we make a copy of the first domain. So, that we can you know delete from that and go back and try again that kind of thing essentially.

So, the original domain D i we retained we work with the copy D i prime essentially. Then for every variable as long as i is between 1 and N, we call a function called select value, which looks at the domain of D i and return some value, which we will call as a i here essentially.

So, what does this function select value do? This function select value keeps looking at values in the domain D i and remember that we are working with it is copy which is called D i prime. Takes out the first value here and these two statements essentially mean that we are

taking out the first value. And, if that value combined with the assignment that has been constructed so far is consistent, then we return the value a i essentially.

If that does not happen and we keep cycling through this y loop till it becomes empty, then we will return null. So, there are two possibilities, either we will return a value a i, which is consistent with the previous variables, when X is equal whenI is equal to 1 there are no previous variables, but as we progress there will be previous variables.

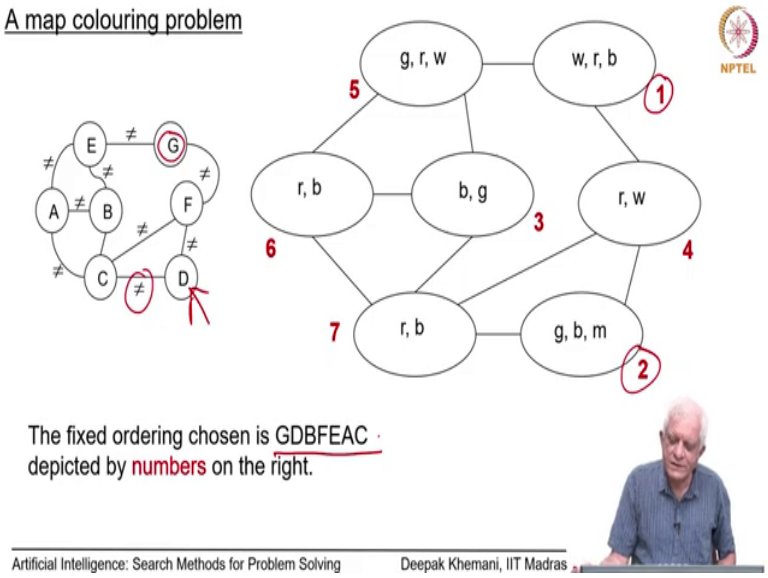
So, either we return a value a i which is consistent with the previous variables or we return null essentially. And, the main algorithm backtracking handles these two cases. If it is null then it backtracks. It goes back to i minus 1 as we see and removes the last value, that we had added to the variable which is i minus 1. So, we will look for a new value for the variable minus 1. So, this is the backtracking part.

If, the value is not null remember we said if a i is null then do all this else we have found a value, then we just simply go onto the next variable, but before that we augment the assignment with this new value that we have found, and then we increment a i. And, now because we are looking at the new variable, we make a copy of that new variable as well essentially.

So, we go through this cycle this is a wide loop. As you can imagine in this cycle I will go up from 1 2 3 4 upwards and if it is backtracking it will come down 4 3 2 1 and so on and so forth. So, it will keep going up and down and as we discussed a short while ago there are two termination criteria's. 1 is that if we have found a solution we we return it else we return null.

So, in our case we are assuming that this assignment is a global variable and in and whether it is null or whether it is non null these are the only two cases either it is null or it has N values in it. And, the N values we reverse just to be consistent with the way people think about solutions that the solution is from X 1 to X N and the values are i 1 to i n. So, that is the simple algorithm backtracking.

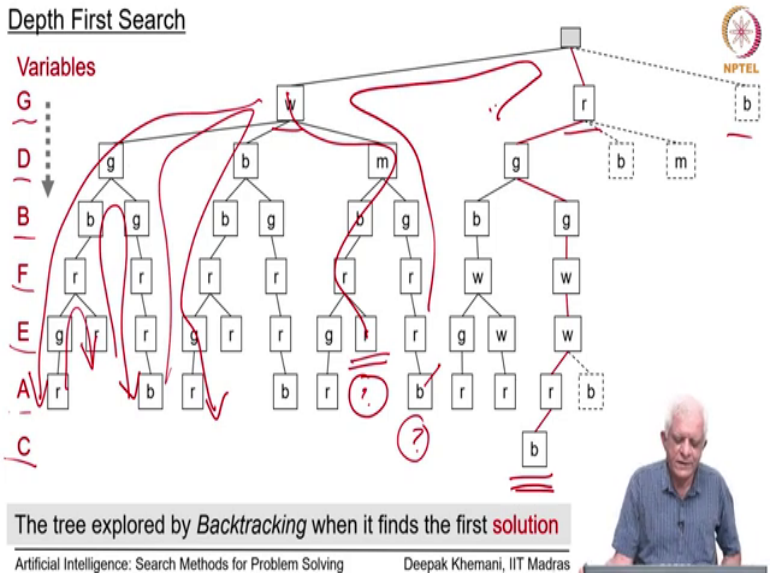
(Refer Slide Time: 11:48)



Let us look at an example here, it is something that we have visited earlier the map colouring problem. In this problem that are these 7 variables A B C D E F G and the relation is not equal to basically which means that you cannot assign the same colour to 2 adjacent nodes.

So, this is just another view of the constraint diagram and what we have stated here is the order? The first node is going to be G, the second node is going to be D and so on essentially. So, we have chosen the order GDBFEAC in some F E A C by some mechanism.

(Refer Slide Time: 12:33)



And, let us see what backtracking does with this? It generates a search tree like this and remember again the order which is given on the left GDBFEAC and remember that we said we will try the first value in the domain, so for G the values were w, r and b.

So, you can go back and have a look at that w, r and sorry here g is here, w, r and b. And, we algorithm tries them in this order w, r and b, then after that it goes to d, it tries a value g for that then so on.

So, essentially it goes on this path then it backtracks and come here, then it backtracks and come here and then it backtracks all the way and come here and so on and so forth. And, you can recognize that this is our good old friend that first search algorithm. And, it keeps doing

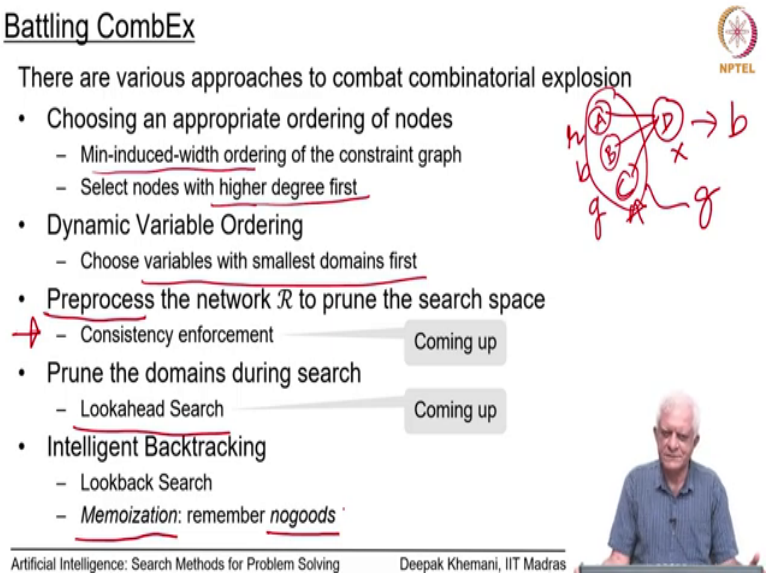
that till it has found here a solution, where it has found the value for all the 7 variables essentially.

So, that is the simple algorithm backtracking, that we will start with essentially. And, as we can see it basically does in that first search. Unlike some of the other things that we had studied, it does not have to go till the end before backtracking. So, here for example, it can backtrack from this level itself, because you could not find a consistent value for the variable a which is consistent with the partial assignment that we had constructed till here essentially.

So, it backtracks and tries something else essentially. There in the next cycle it does find a value for a, but it cannot find a value for c. So, it backtracks all the way there and goes here and then comes here and eventually finds the solution. So, that is our backtracking algorithm.

Now, the thing to observe is of course, that this is the problem that we have been facing throughout this course. That we wade through a search tree and we run into this problem of exponentially growing trees, or exponential search, or combinatorial explosion, and our whole effort has been in this course to try and fight that, how to get around this exponential algorithms?

(Refer Slide Time: 14:57)



We gave it a name we call it combex. Now, there are various approaches that the community tries and it is a growing in a large community. One approach is to choose the nodes in an appropriate order.

How can it decide which order is best? One thing is if you look at the degree of every node essentially. Now, if you choose a node with 1 degree first. So, let us look at a small tree network. Let us say that it is a map colouring problem and it has got these 3 colours rbg, which can be used to colour these 4 regions.

If, you choose the 1 degree node first, so, here or there is 3 1 degree nodes here. If you say that this is going to be r, and this is going to be b, and this is going to be g, then you can see that there is no value that you can choose for the fourth node essentially.

So, the degree of a node matters on the other hand if you started with the high degree node first um. So, let us give them names. So, if you try A B C, and then you try D, then you fail, but if you try D first, then in fact, you can see that just two colours are enough. Let us say, you colour D with b and then all 3 then you can colour with g essentially. With just 2 colours you can colour the whole map and the difference was in the order in which you tried the variables.

If, you try the variables with higher degree first then you are more likely to succeed. By this we mean that you do not have to backtrack of course, even if you tried A B C in that order, by the time you finish A B C and you reach D, you will reach a dead end and you will backtrack. And, you will say let us try something else for C. So, you could try b or r and then you would find the solution.

But, if you try the higher degree node first, then this backtracking is not required. So, the whole effort is to try and reduce the amount of backtracking that a search algorithm does and 1 way to do that is to find ordering of nodes.

So, that the higher degree nodes come first and there are these standard graph algorithms min induced width ordering, which we will not go into the details, but that is just the general idea is that, higher degree node should be tried first essentially. So, that is choosing an appropriate ordering of the nodes essentially.

We can also choose the ordering of variables, by looking at the domains of the different variables. Now, and I extreme example again is that, if there is some variable X i, whose domain has only 1 value. Then, you do not have any other options. So, you might have freeze that value first and put xi as the first node that you will see.

So, this other heuristic is that you choose variables with the smallest domains first. So, that they are not constrained by other nodes, which have larger options available, but they constrain the other nodes. In they are already constrained, because of the number of values are

small. And, then of course, the other nodes will have more options and so maybe hopefully you will find the solution without backtracking. So, that is called dynamic variable ordering.

Then, there is an option to pre process the network. So, that we prune the search space essentially this is called consistency enforcement. And, this is what we will look at next I think. There is still another option which is to prune the domains while doing search.

So, in this option here, we are saying that pre process the network and then do search. If, you were to study constraint processing in a little bit more detail, you will realize that this pre processing sometimes does extra work.

So, people have said that instead of pre processing the network, why do not we do this consistency enforcement, while you are searching. And, in fact, that is the next method that we will look at and these methods are called look ahead search. So, these two things we will study in this course here.

But, there is also this very interesting idea of intelligent backtracking. Essentially, we will mention it in the passing, but the basic idea of intelligent backtracking is this is that, if you have reached a dead end. Let us say in the Ith node, backtracking algorithm says go back to i minus 1 and try a different value essentially.

But, that may not be very effective and the reason for that could be that, the inconsistency for the Ith node is not caused by the i minus first node, but by a node which came much earlier essentially. So, intelligent backtracking would say go back and change the value of that node.

We will not have time to go through this in this course, but we will again come back to this and make a passing mention to that. And, another thing we used to fight this combex monster is something called memorization and the idea of memorization is to remember certain things which do not work essentially.

So, at some point when you reach a dead end, you remember that this combination of values of this assignment cannot lead to a solution. So, that if you keep a memory of such things, which is done by a process called memorization, you keep a memory of what are called nogoods? So, nogoods are parcel assignments which cannot be extended to a solution then if you are backtracking and you reach a nogood then you do not even try something you just go back and try something else.

So, essentially there are these 2 or 3 different ways in which people try to find this, find this combinatorial explosion 1 is to choose an appropriate ordering of the nodes. The other is the pre processor node to do consistency enforcement. The third is to do consistency enforcement while you are searching and the fourth are called lookback methods, which we will not have time to go into.

So, the next thing that we will do is to look at consistency enforcement and we will do that in the next session. So, see you then.