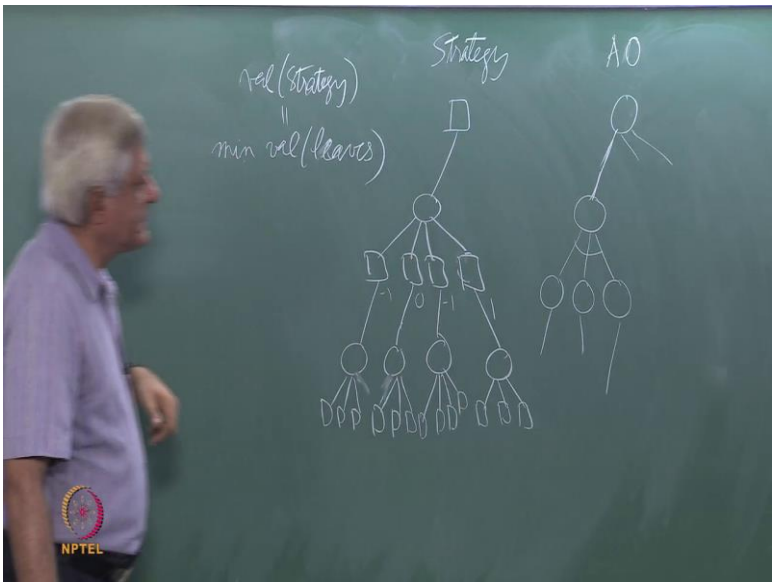
Artificial Intelligence
Prof. Deepak Khemani
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture - 27
Game Playing Minimax Search

(Refer Slide Time: 00:14)



So, let us continue with a study of game playing, and we saw in the last class the notion of a strategy, and a strategies constructed by making one choice for max, considering all choices for min, and then one choice for max, and then all choices for min. So, you can say that solving a game tree, means discovering an optimal strategy we said, that the value of a strategy, is minimum of value of leaves. Assuming that, we are talking in this minus 1 0 1 valuation, where 1 stands for win for max, 0 stands for draw, and minus 1 stands for loss for max. And if once maxes shows on the strategy, then the value of the strategy is the lowest value, from the leaves in the strategy, remember the strategy is a sub tree, and what are the value that the leaves are to minimum of those is what the value of the game will be. And assuming that the opponent is perfect and we assuming that the opponent is perfect, my feel look at and and over graph.

We viewed our solution and and over graph, if you look at what on solution is, at all level you have to choose one, we have to make one choice, and at end level we have to make all choices. So, if for example, if they you had three choices here, then at all level

you will choose one choice, to solution would be like this, and if this was an hand node, you would have to solve for all three, and then if this was an all node, when you would again make a one choice essentially.

So, we can see a similarity between and over trees and game trees, in and in and over trees, if you added and node, you had to solve all the three, all the sub problems emanating from there. If you had an all choice, you could choose any one of them essentially. So, a max node is like an all choice, because max can choose anyone essentially. But max has to en counter for all of min choices, so a min node is like an and node essentially, the differences that the value of the solution, is in this case a some of the values of cost of these three nodes, solving this nodes.

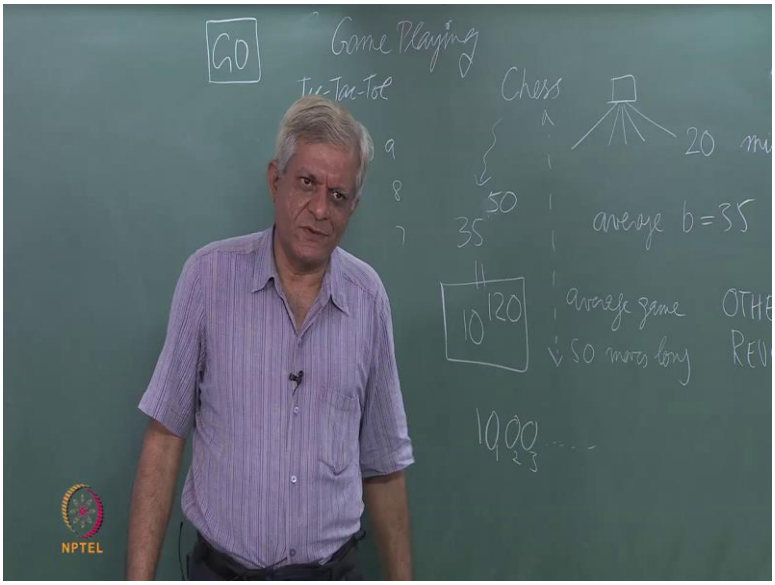
In this case, it is the minimum of the cost of these nodes essentially, where it is a min nodes sitting here, if this was minus 1, this was 0, this was minus 1, and this was 1, then min with select minus 1, so it is a minimum of these value essentially. You can see that, so also consistent with our logical notion of, and always choose is the minimum. So, a and b, if a (s) falls and the whole thing becomes falls essentially, so and also in some sense chooses the minimum value or chooses the maximum value essentially.

So, there is clearly an analogy, between and over trees and game trees; of course, game trees have a very well defined layer structure, which we can always impose on some and over problems, but we have an algorithm for solving and over trees, which is the A O star algorithm. So, my question is can we use the A O star algorithm to solve the game tree, what do you want, we want a strategy of optimal value, in this case a roundly three possible values 1 0 minus 1, so if there is a winning strategy we want it. Otherwise we want the strategy it will draw the game, otherwise we are force to accept a loosing strategy.

So, can we use the A O star algorithm here, there is also there is analogy, in the and over graph the solve nodes are nodes which have a label attached to them, and leaves also have a label attached to them. In some sense, the similar back up procedure is taking place in A O star, if you remember algorithm essentially, in the forward phase. We expand the search a little bit, and the backward phase we backup the values, which is

very similar to the backup values that we using here. Except the of course, here we I choosing min and max, there we I just summing up the values essentially. So, the answer to this is actually yes, we can use the A O star algorithm, provide it we have access to the complete game tree.

(Refer Slide Time: 05:37)



And why is that not always a feasible thing, let us look at the game of chess, which is been the game, which has fascinated most programmers or computer scientist essentially. Now, if you count the size of a tree, as the number of leaves, which means, each leaf is a different game that you can play. Every pass that you can take in the game tree is a game, and the leaf is outcome of the game essentially. So, the number of leaves is in some sense a measure of, the size of the game tree essentially.

So, let us try an estimate, how could the tic tac to ((Refer Time: 06:22)) not is a simple computationally a simple game, because at one level you have 9 choices, assuming that you could you cannot, distinguish between symmetric situations let see, you have 9 choices, you have then 8 choices, at the next level 7 choices and so on. It is a very, not a very large game tree essentially, which is not surprising, because we have figure out by now, that it is the value of the tree is a draw essentially, what about the chess game tree?

As I said, at a max first level, there are 20 choices, those of you know chess will agree with me, eight pawns eight pawn moves with one step, another eight pawn moves with two steps, and four knight moves for the two knight essentially. So, 20 at the first level and as you start playing the game, the game board opens up, and more moves are possible. So, for example, bishops can start moving, cooks can start moving, the queen can start moving, and the number of the balancing factor in some sense increases, as you go towards a middle game essentially.

So, chess players tend to categorize the game into opening, middle and the end game. So, towards a middle game, it is a most complex part of the game. So, people say that on the average b is equal to 35, that on the average, a chess game has 35 moves, that a player has a choose from essentially. And on the average, a typical game is 50 moves long, so average game of course, some games get over earlier some games can long last and so on. But just to get an idea of the size of this game tree, so we can say that, you can one big estimate is set the number of games possible is 35 raise to 50.

So, if the punching factor was concert 25, then you can 35, then you can see, the top level 35 moves and another 35 moves for each of these moves and so on. So, since it is 50 moves long 35 raise to 50, which is equal to what 10 raise to 120, which means of course, just to remind the point, if I started writing this, and I label the zeros, this is the first 0, this is the second 0, this is the third 0, and I will have to keep writing to the end of the board, till I write the one twentieth 0 essentially.

We have already discussed this kind of numbers before, that this is not the number, you can refill with, there is no hope ever of solving the game tree completely, chess tree completely. And in fact, we do not we have an been able to solve it, still now even though people, putting massive computing power towards solving game trees, and they are using very sophisticated techniques like, analyzing the end games, and creating of look up tables for them and then using those to, so all the games trees and that kind of stuff.

Checkers incidentally I do not know, if you remember when you went through this progress in computer science, at some time in this century 2005 or something, I do not

remember exactly, the game of checkers was solved. So, it was known, what the outcome of the game is, and this was done through use of massive amount of computing power, and checkers as the much smaller game, because the choices available for each piece are very limited essentially.

So, if you know the game, you can only move in one or two directions, one step or sometimes you can jump over other than so on, is a much smaller game in terms of the size of the game tree. And that we have just minus to ((Refer Time: 10:38)) chess is practically impossible, if you remember the kind of numbers you are talking about, 10 raise to 75 is the number of fundamental particles in the universe. And then, if the each of them was a super computer, so we can do that argument again, and you can see that you can, any fact not solves the game at all.

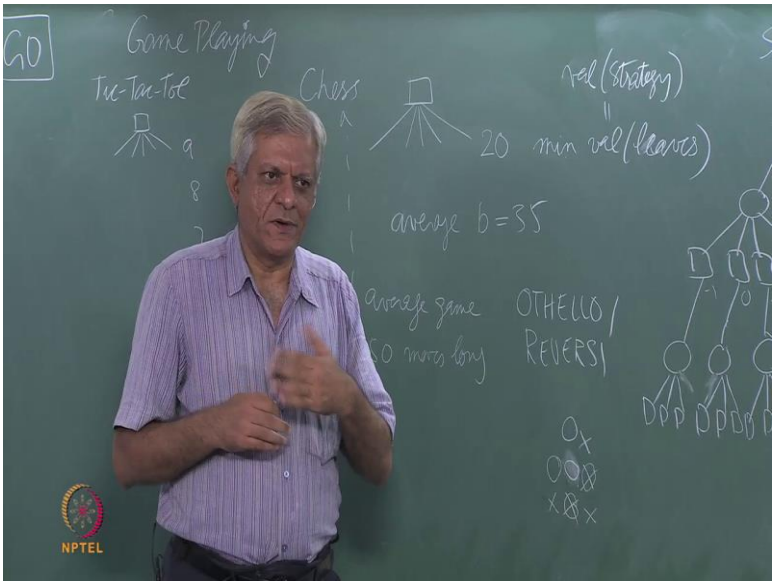
So, we do not know, where a why it will always been in chess or not, which is by the game is still interesting, as oppose to cross and nodes, which we do not want to play because we know that it is always a drawn game essentially. So, if we do not have access to the leaf then how can we use this algorithm, we cannot use the algorithm. So, we have to do some other approximation, and which is what we will look at, essentially is that kind of variations of ((Refer Time: 11:24))

While I am talking about games, the most complex game, in terms of size of the game tree, is the game called go, I do not how many of you are heard about it, it is a game which is extremely popular in Japan, and size of the... and a feeling of the complexity of the game, can be got by the fact that it is plays on a ninteen by ninteen board, and you can plays, it is a game in which you plays coins, on one on top of the, not one. So, their specified locations at cells you can place coins, and you can imagine that in the first move, we can place it an any those ninteen by ninteen locations and then so on, so it is a huge game in that terms.

Go is something that we have not been able to program well, by view in the whole computing community, has not been able to produce good go playing programs. And the people who talk about go; they say that no you have to use other techniques like pattern recognition. By pattern recognition, I mean you know, trying to make out which both

positions are good, and which both positions are not good essentially. Some people use the word zen with go and so on. So, games can be quite come, so go is much more complex then chess in terms of, the size of the game tree which and be can really not too much with that essentially.

(Refer Slide Time: 13:10)



One game that, is of interest to us, is a game call othello, also called as reversi, and we have been using that for the last few years, for the game competition and will do so this year also. So, maybe you should go and look up the game, the idea of the game is that, it is played on a chess board, like board, except that it is a single color board, and we start of by playing, two kinds of coins. So, which let me say, I am representing by knots and crosses like that game, but it practices like, red coin and white coin, and things like that.

So, let say cross stands for red coin, and this stands for white coin, this is the initial position of the game, in the center of the board and a move, you a move is made by capturing opponent pieces, which means that, if I have a piece coin here, I can place the coin here. And in the process, I will capture this piece essentially, which means this piece becomes actually mike piece. So, I have made one move, and this is a game, then it is opponents turn, so opponent can also do something similar. So, opponent can for example, put a coin here, and then this is captured back, in some sense, then I can place a

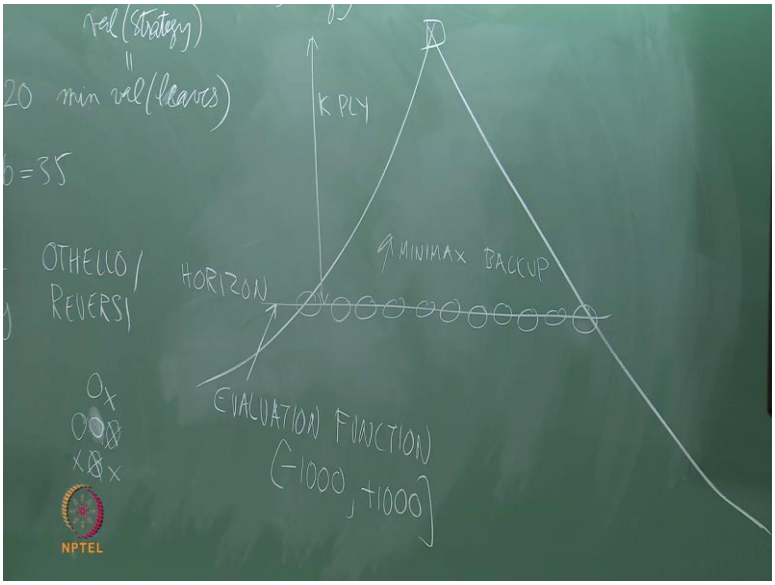
coin here, and capture this back again and again, and the game sort of goes on.

So, if I, if the opponent were to put a coin here, then I then you would simultaneously capture this and this essentially. So, both will get captured, because by placing a coin here, you are enclosing one end you have this, and this end you have this, and this end you have this. So, both in both directions you can... so in all... So, you can captured in four directions, and if you can make multiple captures in one move, but you can, you allow to place only one coin on the board.

And essentially, the game ends, when either one side cannot make a move or every or all the coins have been place on the board. So, it is not uncommon for the board to be almost completely fill them, and you win the game, if you have more coins in the board at the end of it, but we have will be also give you a score has to how many points we have, one by essentially. So, if it is a massive win. So, let say eight by eight board, you have sixty point coins, and opponent is four coins, then you have a big win essentially.

Whereas, if you have 30 and opponent has 34, then he has a smaller win. So, that is the game that we will use for the competition, you will have to write a program, which will compete against other peoples programs. So, how do we play a game, whose search whose tree we cannot access essentially? So, we do what. What humans do is that we do a partial look ahead.

(Refer Slide Time: 16:35)



So, in most game playing programs, you do a limited look ahead, which means, that if my tree looks like this, this is max, then instead of search, so I have drawn with this as a practically, and infinite tree growing exponentially, we cannot explore this threes. So, what do we do we, cut off at some level, this some people called as a horizon, we cannot see beyond that, and this is called a number of plies. So, this is k ply, if the ((Refer Time: 17:15)) So, instead of searching the entire game tree, I mean if you have the complete game tree, then you could have just pack up the values, and you would have found the minimax value, and the best move for max in the process.

But we do not; we cannot search the game tree, because we have seen that they can be really huge. So, we decide that we will low or limited look ahead, and try to decide what is the move. Now, incidentally, that is how human beings also tend to play, at least the beginners, that is how they tend to play, they do a limited look ahead, may be two moves ahead or three moves ahead, but they do not do a complete look ahead, in the sense they do not look at all possibilities.

So, most chess players for example, would not even consider all the twenty moves that you can make at the starting point, they would have a fancy for two or three different possible moves. So, either this pawn opening, or that pawn opening, or something like

that, and they would only explore that. So, we do a limited search, in the sense that we do not look at the complete branching factor, we only look at the few possible moves, and a few possible replies by the opponent, and a few possible moves that I can make the, and in some sense of a search is incomplete essentially.

Of course, expert chess players tend to do more exhaustive search, they also tend to be able to judge positions. So, let us, let me introduce that idea at this movement, then if you are doing a limited look ahead, what is the use, because the nodes that you are going to look at this level. They are not completed games they are sort of half way plate games, but you have made a few moves, opponent as made a few moves, and based on this look ahead, you want to decide, what you want to play essentially. So, what do we do, we have to write algorithms, to go values to this nodes essentially, so what do we do, we apply evaluation function.

So, again those of you might have played chess, sometimes or watch other people play chess, then you can hear comments like, this is the good position for white or something like that or white is winning. So, white is not yet one, but you look at it in say it looks like white is winning, instead of making such qualitative statements, we want to give quantitative values. So, we want to define a function which is an evaluation function, which will give us a value for the board position. Now, what do we do instead of this, 1 0 minus 1 those three values, which are available to us, when the game ends, instead of that we break it up into a larger range.

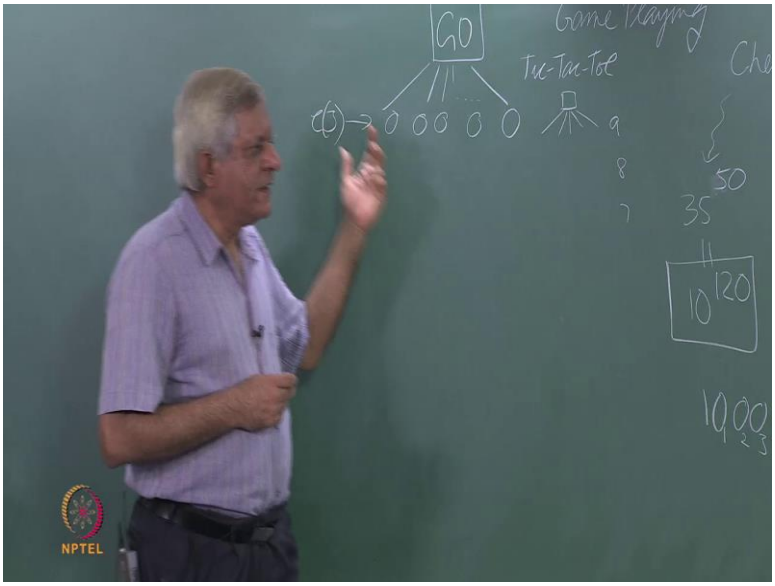
So, typically it is, let say minus 1000 to plus 1000, so the evaluation function returns as a value in this range, which the understanding that plus 1000 is equal to the original one, minus thousand is equal to the original minus, one and all values in 0 is equal to the original 0, which means both sides are roughly equal, but the actual number tells you, how good it is for max or how bad it is for max. So, if it is plus seventy, it is a little good for max, if it is plus 700, it is much better for a max, and if it is minus 600, it is quite bad for max. So, we try to look at a board position, and give it a value, give it a number.

Now; obviously, this means, different kind of reasoning, and in some science it involves the use of knowledge, about the game essentially. So, you should be able to look at a

game, board position and give a number essentially. So, if we can now apply the evaluation function to each of these nodes, on the horizon, which is wherever search ends; then we can apply the same minimax back up rule, to evaluate the value of this game.

So, the backup rule let we said, that at min level, choose the lowest value from it is successive, and at max level, choose the highest value from it successive. So, we can back up these values, and determine what is the value for that, and in the process also decide what is the best move to make essentially. Now, clearly the performance of this algorithm will depend upon how good your evaluation function is, essentially, because if your evaluation function is good, then it will tell you which of the board positions are better.

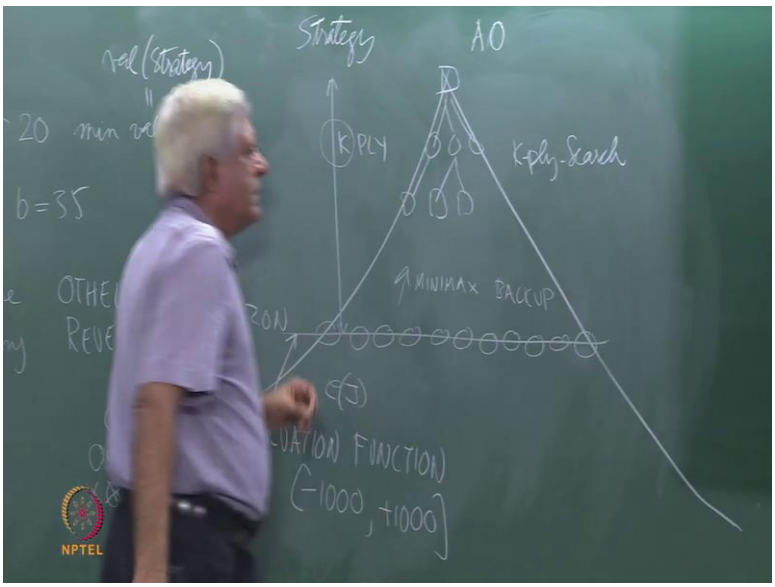
(Refer Slide Time: 23:02)



Now, ideally we would like to have a perfect evaluation function, which in a game like GO people have try to do, is that you look at all the choices, and apply the evaluation function here. So, we will call this function e, and let say e of J, were J is a node. So, we apply e of J here, ideally we say just look at the all the options and see which one leads to a better move essentially.

And then make them, but in practice, evaluation functions are never perfect, they are like heuristic functions, you know you are making some judgment, and arriving at some number that may not be accurate. So, in practice, it has been observed that a combination of evaluation and look ahead does reasonably well for games. So, this is the situation where a there is no look ahead, except that at this level, you just look at the choices, we have and picking the bests base on a evaluation function.

(Refer Slide Time: 23:59)



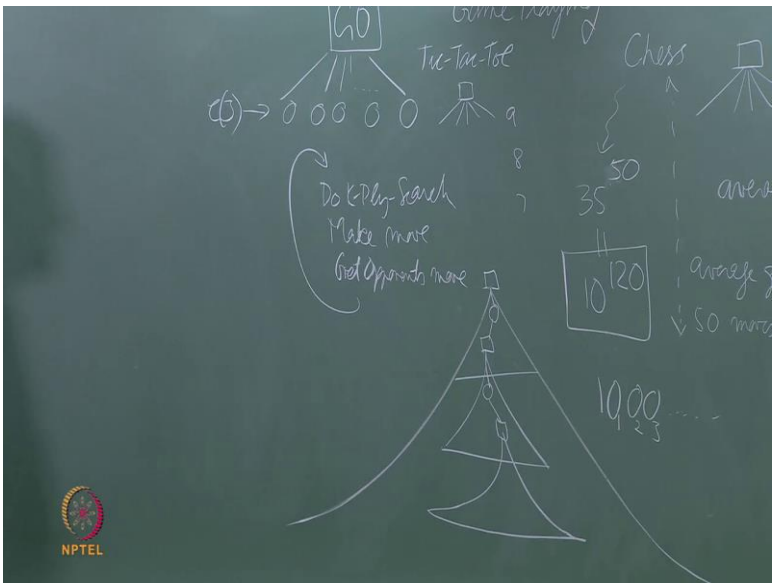
Here, you are saying, I will look at my choices, first and I will look at what the opponent can place, in that position and so on. And I will keep doing that, to some k ply, depending on how much computing power I have, go up to the k (s) ply, apply the evaluation function there, and then back up the values essentially. So, in some sense, what will happen is that, that is something will get captured in the look ahead.

So, again if I use a analogy of chess, then if you are doing, you are capturing the opponents look, and the opponents is capturing you bishops or something than at least those things are known, that this, this is are going away essentially. So, the absence of an evaluation function is compensated by doing more look ahead essentially, typical chess playing programs that you get, on laptops for example, would do something like eight ply or something like that. And generally, you can imagine that with this kind of

branching factor, the number of games that you have to look at is growing exponentially.

And people have surmised, that if you do sixteen ply look ahead for chess, with risibly good evaluation function, then you can play the grandmaster level essentially. So, it is a... so nice in practice of course, we do not do such simple searches, we will see, that sometimes we do a little bit more search in some areas and so on and so far. So, what is the game playing algorithm, we want to write, see we want to still win the game; we do not want to make some move and say it is a good move essentially. So, the game playing algorithm that we will use is. So, let us just call this k ply search.

(Refer Slide Time: 26:10)

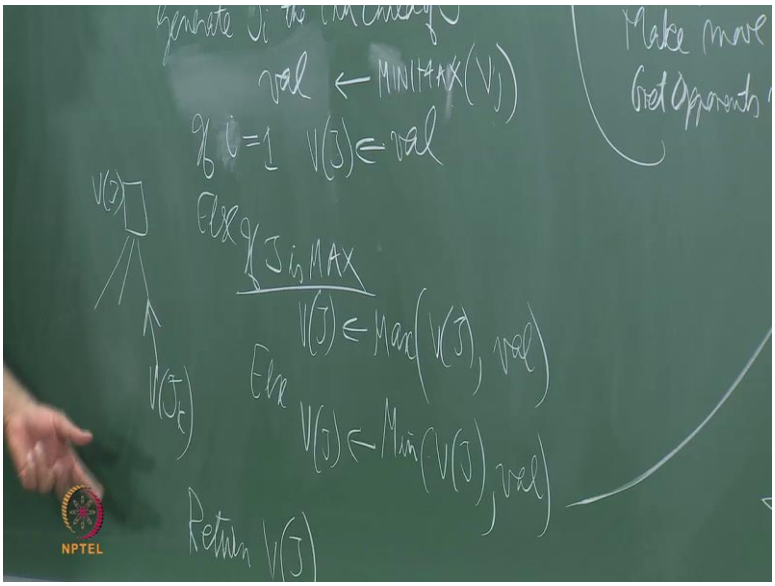


We will have an do in which will say, do k ply search, and then make a move, and then get opponents move, and we will put this into a loop, till the game ends. So, what are we doing now? If the game tree was something that we can explore completely, we would have analyze the whole game tree once for all, and said this is the strategy, and it is a winning strategy. And then, you just play it according to that strategy, but the game tree, we are not able to access and what we are doing now, in this algorithm, is that at every move you are doing a search, every time you have to make a move, you do some search, limited search k ply search, and then decide based on that.

So, what does this amount to? This amount to saying that if you make a choice here, let say this is your game tree, and you do a search, let say up to this ply depth, which means you make a move based on this much search. So, this is your move; then you wait for opponent move, so opponent makes a move, let say opponent makes this move, so here. Somewhere inside that original search that you did, now I this stage, you do another search, limited search, again k ply search, and then make your second move. So, let say this is your second move; then opponent makes a move; then again you do a search.

So, in this manner, you can see that for every move that you are making, you are doing a limited search essentially, what is a advantage of that, is that there is a game and force you are looking at those part of the three, which you are not seen originally. Originally you have seen only till this part of the tree, and then after two moves, you can see two plies deeper. Because you are made one move and opponent is made one move, then your search will now look a little bit deeper, then again here, another two plies deeper and so on. So, as it goes along, you are looking at different, so all that remains to do is to write this k ply search algorithm that is the simplest of game playing algorithms.

(Refer Slide Time: 29:12)



We will see, how to improve upon that in that following classes, so let we first write the algorithm, it is a very simple algorithm minimax, it is call minimax, and it takes a node J

as an argument, and it returns the minimax value essentially. So, let us say, this algorithm only computes a minimax value, and on the process, we can put in a small routine, which will tell you what is the best move, that is the secondary thing, which comes out of it essentially. And rough you the algorithm we just follows, to let us assume that you have a way of testing, whether you are on the horizon or not essentially. So, you can have some kind of count, as you go searching into the tree essentially.

So, I am assuming that you will some of figure out how to do that, and then we do the following, if J is terminal, so J is a node, and by terminal we mean a test, which tells you whether you on the horizon or not. If you are on the horizon, then we get V J is equal to e J. So, you simply apply the evaluation function, and you get the value for that note, else it is not terminal for i is equal to 1 to b, where b is the branching factor, generate the J i the i th child of J, then if i equal to 1, which means if you are looking at the first child, then V J or let we uses slightly simpler is this.

Let say something called val is the minimax value of V J, if J, if i is equal to 1, then V J gets val thus the first node, first child that you have looked at. Otherwise you will update to a better child, else if J is max, then V J get max, so I just use this devised, because I do not want write this, this is a recursive call notice, it is a recursive call, with an next node, I just do not know to write it again and again. So, I am just writing in once, so once I making a recursive call, and then if the first one of course that gets the value, V J gets that value, if it is not the first child, then you compare with the current value, and this new value that you are getting.

So, if it is max, it means that, you have got some value here, which is V J, and you have looking at this child, and your getting a value V J, let say k, which is return by this. So, you have, you going to return V J, which is the minimax value of this node J, and so this recursive call will return the minimax value of V J. And then as I scan from left to right, going, I go going from 1 to b, I will keep seeing, if I am getting a better value, from the next call and so on. And wherever I get the better value I will store that.

So, it is a very simple algorithm, which will look ahead k ply deep, and compute the minimax value of that game, based on the evaluation function, because you have at

terminal level, you have applying the evaluation function. And I am assuming that you will augment this with, be able to select, what is the move that max should make, because that is really the task that you have doing, that you do this much search. And this algorithm is basically doing this search here in this area, but you want to make the move also. So, you must keep track a what, where the best move came from essentially.

So, you must keep track of that then you will make the move, wait for opponents move, and then again make a call to V J to decide what is the next move? ((Refer Time: 34:58)) So, this is the simplest of all algorithms essentially, what is the nature of this search can you tell you what kind of search is this doing.

Student: First depth.

It searching this part of the tree, game tree or in if you look at this diagram, it searching this part of the game tree, would in what manner is searching.

Student: Depth bounder, depth first, depth burst.

Depth bounder yes, because we have doing k ply, but within that bound, how is it searching in.

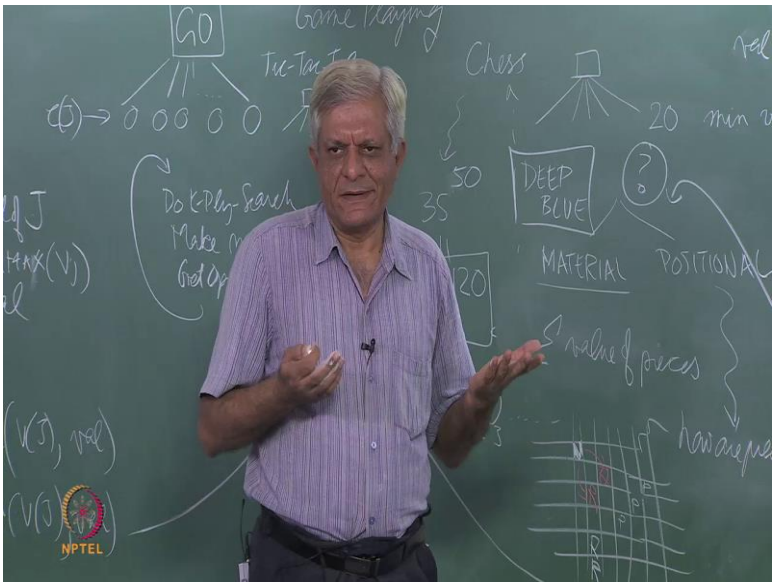
Student: Depth first search

It doing depth first search, so you should figure out, this is really doing depth first search, which of course, give us ((Refer Time: 35:46)) just may not be the best way of looking at things essentially. So, in the next class; of course, we will trying to improve upon this, next may be one or two classes, but the, in the remaining time, today which is about five minutes are so. I want to just spend a little bit of time on this evaluation function, how do you write in evaluation function for a board position.

Because it is the performance of the algorithm really depends on how good the evaluation function is, if it can judge, accurately the value of a board, and by accurately

you mean, whether it is you know how close to winning it is essentially. Then just one play search would be enough essentially, if you have very good evaluation function, but other side in practice it not so easy to get very good evaluation function.

(Refer Slide Time: 36:50)



So, what goes into an evaluation, so essentially you want to look at a board position, let say we are talking about chess, but in practice of course, when you do you will be doing othello, when you look at a goal port position, you want to give it a value between minus 1000 and plus 1000 essentially or minus large and plus large let us see, any ideas how you could give this number.

Student: Any games practice from, wherever it means back tracken keep asign((Refer Time: 37:24)) keep it.

But you see, the whole point of this exercise, is that we are trying to our search problem is so huge, that we cannot search the game tree.

Student: ((Refer Time: 37:39))

How do you play many games?

Student: May be initially begin with random assignment and you will ((Refer Time: 37:48)) and based on once you reach the end, go back track and ((Refer Time: 37:53))

So, I get a point, you are trying to say that, so you are a machine learning enthusiasts and you are saying that, I want to learn the evaluation function, that is that will come. In fact, Samuel Sekars playing program, improved it is game, because it improved it is valuation function on the way, but that comes after word essentially, before that what are the components of the evaluation, I mean.

Student: ((Refer Time: 38:24))

Oil all in machinery terms, what is the structure of a evaluation function, what I am learning, am I learning parameters or weights, weights of what essentially. So, if you look at Tom Michaels book the first chapter, he actually describes, how game playing program can learn evaluation function, but then he gives it a linear combination of, I thing is document checkers, of number of pieces I have, and number of pieces opponent have, number of things I have, number of things opponent has, and a linear combination of them is this thing. So, my question is more fundamental, that if you are to write the evaluation function, just for. Let say you are a chess expert and All right, let say you have Viswanathan Anand sitting next to you, and you say help me write this evaluation function, what could he say.

Student: The answers piece of the pieces will have particular value and then ((Refer Time: 39:22)), and there would be additional advantages, piece advantage.

So, typically an evaluation function will have two components, one is call material and other is called positional. So, chess players will say, white is winning, because white as material advantage, which means you know, white is got let say, one row can one bishop extra. And then any good chess player will say, if you have that much more fighting power, I am not going to play against you or rather you would resigned essentially. So, one thing is material, number of pieces, you can say, some of now beginning chess players might say, that you know a queen as value 9, and bishop has value 5 or 3 or 4 or whatever I do not know, look as value 5.

And then you count, how many pieces, do I have, what is there values, and let say you give negative value to opponents pieces, and from that I sub track how many pieces opponent has. So, if I have more pieces or more valuable pieces, than the opponent, than this, some will become positive essentially. Initially as you can guess, both sides of the same number of pieces a value of material value is 0, both are the same number of pieces, but as you capture some pieces, your material value goes up essentially.

Now, in practice; of course, chess playing people have much final gradation of values. So, they compute in 100, for example, let say bishop is 200, and knight is 220 or something like that, it really depends on your prospective of the game essentially. So, one is the material value, how many pieces I have, and how many pieces opponent has, rather is positional, we says, how are the pieces arranged. Now, this is of course, the trickier part essentially, this is the more difficult part, because it is not looking a structure, and not it is not just a method of counting, how many pieces I have, how are they arranged essentially.

So, chess players, what say things like, if there are two rooks in the same. So, if this is the chess board for example, then if I have a rook here, and if I have a another rook here, in the same column, then chess player say that, it stronger position, two rooks in the same column are very powerful, and you know, you should try to arrive at such a position essentially. All they might say, that you know some pawn structures, linked pawn structures, if pawns are supporting each other, it is better than pawns, if their scattered around the place, and not putting each other essentially.

Then you know that if there is a knight, if n stands for a knight, and if it is the opponent has a queen here, and a rook here. Then you can see that knight is attacking both the queen and the rook at the same time. So, chess plays called is a fork, and a fork is; obviously, a good thing to have essentially, because it among to saying that in the next move, I am going to either capture a queen or a rook. So, you are going to use lose material in the process essentially, because queen and rook are important then knight is not so important in ((Refer time: 43:24)) material value essentially. And there are other things like controlling the center, and attacking the center, and think like that.

Now this program deep blue, and when I read about this is so in 2002 or something like that, it had a 1000 components to positional evaluation. So, just as we said you know, rooks in the same column or connected pawns, a protected king or mobile pieces, so you know bishop is slap then it not very useful, hence things like that. It has a 1000 different component, which were use to evaluate the board position, in the positional part. So, that obviously is the key to the whole thing, if you can look at a board position, and give a value, and this is just an attempt to give an accurate value, it is looking at it piece ((Refer Time: 44:19)) say this pattern is good, this pattern is good, this pattern is good,

And of course, if the opponent has that pattern you are going to subtract it from essentially. So, really the secret is in devising a good evaluation function, if you have good evaluation function, then you do not have to search very much deep in the game, and you evaluation function itself will tell you what is the good position or not essentially. So, what you will need to do, for your othello game is to, look at the game, on the web or read about, what it in try to device an evaluation function, because that is going to be critical part of your program essentially.

So today, we have seen this, that we cannot search the entire game tree, we have to do a limited look up, look ahead, and we have a program to do k ply search. And we will repeatedly call this program, for the first move, for the second move, for the third move, for every move that we make we will call the program. And this simple version of k ply look ahead is essentially doing depth first search, left to right and we want to improve upon that essentially. So, we will do that in the next class.