

Artificial Intelligence: Search Methods for Problem Solving
Prof. Deepak Khemani
Department of Computer Science & Engineering
Indian Institute of Technology, Madras

Chapter - 04
A First Course in Artificial Intelligence
Lecture – 27
Stochastic Local Search: Randomized Algorithms

So, welcome back. Let us continue our journey of trying to escape from the local optima that local search algorithm find themselves in. We have seen a couple of methods of trying to get away from that and we want to look at more methods essentially. Initially, we will look at one more deterministic method and then we will move on to stochastic methods or randomized methods. The need for randomized stochastic method arises because the spaces that we are looking at are so huge that one cannot hope that one can in some you know this systematic manner exclude the entire space.

So, we will see that we want to introduce a tendency of exploration into the search process. So, far we have been relying largely on the heuristic function and we are letting the heuristic function drive the search. We will do that for one more algorithm which is also a deterministic algorithm before moving on to Randomized Algorithms.

(Refer Slide Time: 01:27)

Exploding Search Spaces

N	Candidates in SAT: 2^N	Candidates in TSP: $N!$	Ratio
1	2	1	0.5
2	4	2	0.5
3	8	6	0.75
4	16	24	1.5
5	32	120	3.75
6	64	720	11.25
7	128	5040	39.38
8	256	40,320	157.5
50	1,125,899,906,842,624	$3.041409320 \times 10^{64}$	3×10^{49}
100	$1.267650600 \times 10^{30}$	$9.332621544 \times 10^{157}$	10^{127}



Artificial Intelligence: Search Methods for Problem Solving Deepak Khemani, IIT Madras

Now, the size of the search space is indicated by looking at for example, 2 classical problems which we have been talking about which is SAT or satisfiability problem which we know is an NP complete problem and the traveling salesman problem or TSP which we know is considered by many to be the holy grail of computer science because it is much harder in terms of the size of space that we have to explore.

And this table shows you that as the parameter N increases how do the spaces for these 2 problems increase. N is a number of variables in the case of SAT and it is a number of cities in the case of TSP. So, we have the second column showing the size for the SAT's search space and the third column showing the size for the TSP search space and the fourth column gives you an idea of how much more is TSP as compared to SAT.

And we can see that initially the TSP grows a little bit slower SAT tends to go exponentially which is it starts off with a value of 2, then 4, then 8 doubles every time we add a variable because the number of rows in the table for SAT would double. Initially, TSP is slow it starts off with 1, 2, 6 which is lower than SAT, but as we go beyond 4 TSP over takes SAT and then it grows much much more rapidly. So, if you look at for example, row number 7 which is for 7 variables SAT requires 128 candidates and TSP already needs 5,040 candidates.

As we jump to 50 and 100 two kind of larger set of problems. You can see that for 50 we can still writes SAT as in the normal decimal notation whereas, for TSP we have had to resort to a number like 10 raise to 64 essentially. For the case of 100 for both SAT and this thing the numbers are too small to be written out completely and we have to resort to powers of 10. So, let us see how long will that take for us to solve these problems. So, we will look at SAT for sizes of 50 and 100 and you must keep in mind that TSP is much more complex which is given to us by the third the last row last column in the table.

(Refer Slide Time: 04:11)

Worst case time to solve SAT with 50 variables



N	SAT: 2^N	TSP (brute force): $N!$	Ratio
50	1,125,899,906,842,624	$3.041409320 \times 10^{64}$	3×10^{49}

Inspecting a million nodes a second: ~~1,125,899,906~~ seconds

Assuming a 100 seconds in a minute: ~~11258999~~ minutes

Assuming a 100 minutes in an hour: ~~112589~~ hours

Assuming a 100 hours in a day: 1125 days

Assuming a 1000 days in a year: 1.1 year

For accessing this content for free (no charge), visit : nptel.ac.in

Artificial Intelligence: Search Methods for Problem Solving

Deepak Khemani, IIT Madras



So, for N equal to 50 the number of candidate in SAT is too large for me to read out as a number, but you can observe that it has about 10 raise to 15 or the that candidates. Now, if you were to inspect a million nodes in a second, then we would chop off the last 6 digits from this number and what we would get is the number of seconds that would need to inspect those space for 50 variables.

Assuming that there are 100 seconds in a minute of course, there are not 100 this is a 4 time more conservative estimate 2 and a half times more conservative estimate assuming a 100 seconds in a minute which is a very conservative estimate we chop off 2 more digits from that number and assuming a 100 minutes in an hour we chop off 2 more digits from that number.

Assuming that there are a 100 hours in a day we chop off 2 more digits from that number and we get a figure of 1,125 days which we know is a little bit more than 3 years. But, if you were

to look at 1,000 days in a year then it would come to about 1.1 year. This comparison becomes a much more stark when we move to 100 variables for SAT.

(Refer Slide Time: 05:54)

Worst case time to solve SAT with 100 variables



N	SAT: 2^N	TSP (brute force): $N!$	Ratio
100	$1.267650600 \times 10^{30}$	$9.332621544 \times 10^{157}$	10^{127}

Inspecting a million nodes a second: 1.27×10^{24} seconds

Assuming a 100 seconds in a minute: 1.27×10^{22} minutes

Assuming a 100 minutes in an hour: 1.27×10^{20} hours

Assuming a 100 hours in a day: 1.27×10^{19} days

Assuming a 1000 days in a year: 1.27×10^{16} years

Given a 100 years in a century: 1.27×10^{14} centuries

A hundred thousand trillion centuries!



So, we see that we have a space of about 10 raise to 30 here. For TSP it is much beyond a comprehension which is 10 raise to 157. These are the kind of numbers we cannot even make sense of.

If you were to consider the fact that it is estimated that the total number of fundamental particles in the universe in the entire universe is about 10 raise to 80 or something like that then a number like 10 raise to 157 simply boggles the mind I think. So, anyway we will just keep in mind that TSP is much worse and look at how long would it take for us to solve a SAT problem with 100 variables.

So, if you assume again that we see a million nodes a second then from that 10^{30} we have chopped off 6 6 0s and we are left with 10^{24} seconds. Assuming a 100 seconds a minute we chop off another 2 and we are left with 10^{22} minutes. Another 2 for 100 minutes to an hour and we have 10^{20} it should be 10^{20} hours.

And another 100 hours in a day and this should be 10^{18} days. Assuming a 1,000 days in a year this will be 10^{15} years which is a million billion years essentially and if you count it in centuries you have 10^{13} which is 10,000 trillion centuries. There is a small error in this which you can gloss over.

(Refer Slide Time: 07:58)

Escaping local optima

Given that

it is difficult to define heuristic functions
that are monotonic and well behaved

the alternative is

to look for algorithms that can do better than Hill Climbing.

We look at another deterministic method next,
and will look at randomized methods later

First we look at searching in
the **solution space** or plan space



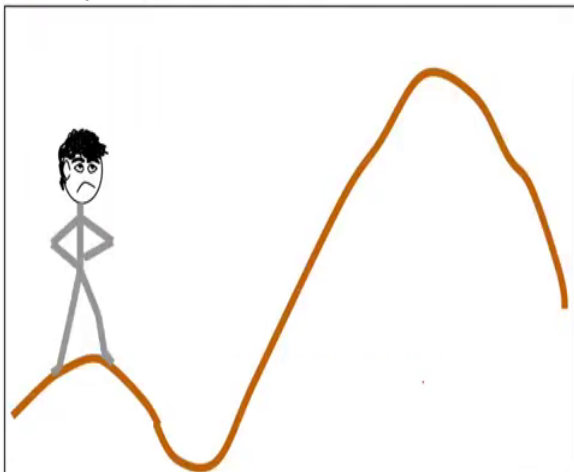
So, our goal was that given that it is difficult to define heuristic functions which behave well which would be nice for algorithms like hill climbing the alternative is to look for algorithms

that can do better than hill climbing. And we look at another deterministic method first. We have already seen a couple and we will move to randomized methods later.

But, what we want to also do is that we want to now introduce this notion of looking into searching in the solution space or what is also called as a plan space. So, this is a space of all possible candidates out of which some of them are solutions and some of them are non and we look at something like a perturbation operator which allows us to move from one candidate to another.

(Refer Slide Time: 08:45)

Local Optima



Artificial Intelligence: Search Methods for Problem Solving Deepak Khemani, IIT Madras

So, let us look at it from the perspective of SAT and our goal just to remind you is to somehow get out of this local optima and head towards the global optima essentially.

(Refer Slide Time: 08:56)

Hill Climbing - a local search algorithm



Move to the best neighbour if it is better, else terminate

```
Hill Climbing
N ← Start
newNode ← best(moveGen (N))
While newNode is better than N Do
  N ← newNode
  newNode ← best(moveGen (N))
endWhile
return N
End
```

In practice sorting is not needed, only the best node

Algorithm Hill Climbing



Just a recap of the hill climbing algorithm. We said that hill climbing is a local search algorithm and it said that moved to the best neighbour if it is better else terminate essentially. In practice, we also observed that sorting is not needed. We did sorting for best first search and we kind of adapted it for hill climbing earlier. But now we can write a version which is without sorting and only selecting the best node at any given point.

So, the algorithm now looks like as follows that you let N be the start node and then you move to the best neighbour of N which is done here. You do the move gen for N and then select the best out of that that you can do by simply scanning the neighbours. And while the new node is better than N it better would be higher in the case of maximization lower in the case of minimization, but we will just use a generic term better. So, while the new node is

better you move to the new node which is what is done here and then generate the next best neighbour and as long as that is remains better we keep moving to that essentially.

(Refer Slide Time: 10:08)

Best Neighbour-a variation of Hill Climbing



```
Best Neighbour
N ← Start
bestSeen ← N
Until some termination criterion
  N ← best(moveGen (N))
  IF N better than bestSeen
    bestSeen ← N
return bestSeen
End
```

Move to the best neighbour anyway

Algorithm needs external criterion to terminate



So, let us look a va[riation] look at a variation of hill climbing. What we want to say is that move to the best neighbour anyway. This is different from what we did earlier. In hill climbing, we said move to the best neighbour if it is better than the current node. Now, we are saying we do not have this criteria of being better than the current node you move to the next neighbour anyway.

So, the algorithms become becomes a much simpler. We start by saying that N is a start node and since we are now going to be moving around the space without terminating at a maxima we want to keep track of what is the best that you have seen and we will store it in a node called best seen essentially.

When, we initialize that to the node that we start with and then based on some termination criteria. This termination criteria will typically say that do for so many rounds or something like that or you could say that till the best seen is not changing over a 100 cycles or 1000 cycles or something like that. So, some such criteria which tells you when to stop you simply keep moving to the best neighbour. Irrespective of whether it is better than the current node or not essentially.

But, what you do is to keep track of the best. So, that whenever you see a better node you remember that this was the best node you have found and in the end you return the best seen node that you have seen throughout your journeys here ok. So, as we said it needs some external criterion to terminate which will typically be based on how much computing power you have essentially ok.

(Refer Slide Time: 11:44)

Getting off a Local Optima

Can Best Neighbour find a path to a better optimum?
Algorithm Best Neighbour *does* get off the optimum
BUT
Moves right back in the next cycle!

NPTEL

Artificial Intelligence: Search Methods for Problem Solving Deepak Khemani, IIT Madras

So, our goal is to get off this local maxima and we have kind of try to achieve it by saying that move to a neighbour move to the best neighbour even if it is not better than the current node. So, as you can see our goal is to somehow get off this small local hill or local maximum and head towards the global maximum. So, the best neighbour algorithm that we just wrote can it find a path to the optimum. As you can see it does get off the optimum because we have not put the criteria that the neighbour should be better than the current node.

So, it can even be worse and as depicted here you are moving from the top of the local maximum to a node which is nearby, but which is a little bit lower. The problem with this algorithm is that in the next cycle when it looks for the neighbour of the new current node it will just move back right back to the local maximum because that is the best neighbour for the new node essentially. So, this is what would happen and this is what we have to somehow stop happening and which is what we do in this algorithm which we called as tabu search ok.

(Refer Slide Time: 13:09)

Tabu Search – not allowed back immediately



```
Best Neighbour
N ← Start
bestSeen ← N
Until some termination criterion
  N ← best(allowed(moveGen (N)))
  IF N better than bestSeen
    bestSeen ← N
return bestSeen
End
```

Move to the best neighbour anyway,
but only if it is not tabu

taboo

The word *tabu* comes from the Tongan word to indicate things that cannot be touched because they are sacred. - *Wikipedia*



So, the basic idea in tabu search is to allow the search to keep going to the best neighbour, but make sure that you do not turn back immediately from wherever you have come essentially. So, we classify certain nodes as tabu which means that you are not allowed to go back there and you move to that best neighbour which is not classified as tabu. So, we can write it as saying that move to the best allowed neighbour. So, move gen gives you the set of neighbours and allowed tells you which of them you are allowed to move to and move to the best of them.

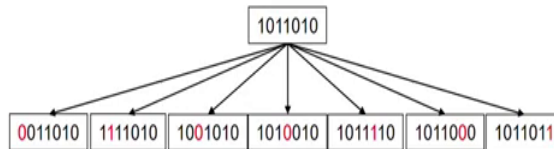
And as before if you find a better node you keep track of that and eventually report the best node that you have visited. So, the word tabu comes from the tongan word which indicates the things that cannot be touched because they are sacred in some sense. I got this definition from Wikipedia. It is similar to the English language word taboo it just said the spelling is a little bit different essentially.

(Refer Slide Time: 14:30)

Tabu Search for SAT



When $N=7$, the neighbourhood function N_i changes ONE bit.



Maintain a memory array M that keeps track of how many moves ago a bit was changed

Initially $M = 0000000$

A *tabu tenure* tt defines the window within which any bit cannot be changed



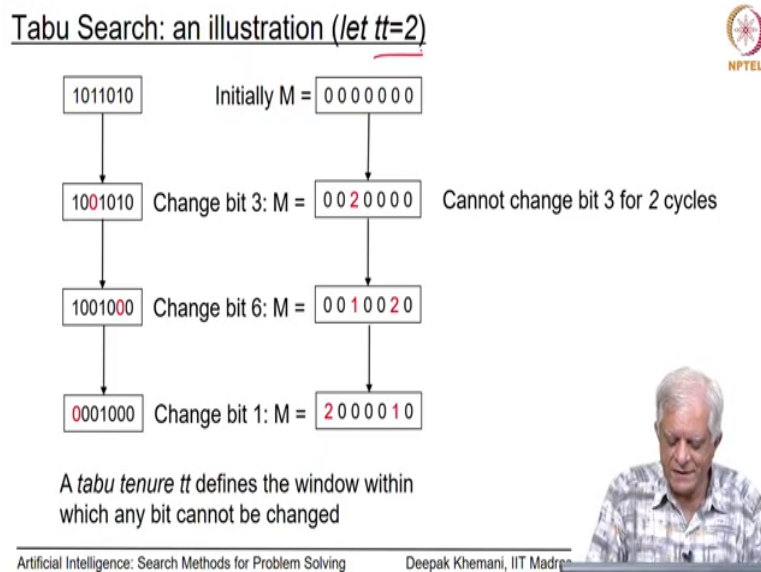
So, if you wanted to do tabu search for SAT then we have seen what does this state space search look like. If you had 7 variable SAT problem and if you chose a neighbourhood function which we had called as N_1 which says that you can change one bit then you would have had these 7 neighbours and the numbers in the red show the which the bit which has been changed essentially.

So, what we do in tabu search is to keep track of which bits have been changed. So, we use a memory array to do that and there are various ways of representing this, but we use an array called M that keeps track of how many moves ago a bit was changed essentially.

And we initialize the array to 0 which says that essentially what this array M tells you is as to for how many moves you cannot change a particular bit essentially. And when you start off M is equal to 0 which means you can change all the bits, but as we change bits we will kind of

mark them as saying that no you cannot change this bit in the near future. This notion of near future is captured by something called tabu tenure which we often use of variable tt to denote that and it defines a window within which any bit cannot be changed. So, I will just illustrate this with an example here.

(Refer Slide Time: 16:09)



So, let us say that we are looking at this 7 variable SAT problem and we have assumed that the tabu tenure is 2. What does that mean? It means that if you have change the bit then for the next 2 cycles you cannot touch that bit now you have to change some other bit essentially.

So, let us say that this is a initial candidate the 7 bits are 1 0 1 1 0 1 0 and initially array M contains all 0s. Now, let us say we changed the 3rd bit and arrive at a new candidate where red shows that you have changed the 3rd bit. So, the new candidate is 1 0 0 1 0 1 0 and we assume that this is the best of the 7 neighbours that the starting m bit had.

Now, in the memory array M we changed the third value to 2 and this signifies that you cannot change this bit for 2 cycles essentially. So we move on further. So, let us say we move to the next best candidate which will be one of the 6 candidates that you can one of the 6 bits that you are allowed to change. And let us say for arguments sake that you changed the 6th bit.

So, the 6th bit has it been changed from 1 to 0 as you can see here and the memory array has been modified. So, the value for the 3rd bit has come down from 2 to 1 which means that you cannot change it for one more cycle whereas a value for the 6 bit has now become 2 and which says that you cannot change it for 2 cycles.

So, you can see that that if the tabu tenure is 2 then in the steady state as the algorithm has move forward a few steps there would always be 2 bits that you are not allowed to touch at any given point of time and those 2 bits will be the last 2 bits that you had changed in the recent past essentially.

So, if in the next stage for example, you were to change the first bit then the first bit becomes tabu for 2 rounds and the sixth bit is now tabu only for one round as we can see in this figure. And the 3rd bit has value has become equal to 0 which means now you can go and change it again. So, the tabu tenure of 2 set that if you have changed a bit you cannot change it for 2 more rounds essentially and that is a simple algorithm.

There are other ways you can try to implement this notion of tabu. For example, you could adapt the best first search algorithm and keep a finite close list. So, you keep a circular close list which means that only let us say the size is k where k would be the tabu tenure then the last k nodes that you have seen they would always be in the close list and you are not allowed to move to them. So, there are different ways of doing it, but the basic idea behind tabu search is that if you have moved off from maximum then do not go back to the maximum immediately and there are different ways of implementing that essentially.

(Refer Slide Time: 19:17)

Tabu Search – ASPIRATION criterion



```
Best Neighbour
N ← Start
bestSeen ← N
Until some termination criterion
  N ← best(allowed(moveGen (N)))
  IF N better than bestSeen
    bestSeen ← N
return bestSeen
End
```

Move to the best neighbour anyway,
but only if it is not tabu

If a *tabu* move results in a node that is better than bestSeen then make an exception for it (that is, add it to allowed set).



Sometimes what may happen is that one of the tabu moves is actually a very good one essentially. What do we do in that case essentially? So, what we say is that there is a certain aspiration criteria and the aspiration criteria says that if a tabu move results in a node that is better than the best seen that you are never seen such a good node in the past and the only reason you cannot move to that node is because that move is been put in the tabu list, then you can make an exception for that move and add it to the allowed list. So, this is called the aspiration criteria which says that opportunistically if you see a very good candidate.

(Refer Slide Time: 20:08)

Tabu Search: driving search into newer areas



Maintain a frequency array F that keeps track of how many times a bit was changed

Initially F = 0000000

Let F = $\overset{100}{F_1} \overset{212}{F_2} \overset{5}{F_3} F_4 F_5 F_6 F_7$

Penalize each move by a factor proportional to frequency. Let node_i be generated by changing the ith bit

$$\text{eval}(\text{node}_i) \leftarrow \text{eval}(\text{node}_i) - F_i$$

- for a maximization problem



Another approach that many people use is that if you want to drive search into new areas then if there are certain moves that you have not made often enough. And by moves in this SATs problem we mean bits set you have changed, then you try to make those moves or try to encourage the algorithm to make those moves more often.

So we do this by maintaining an array which is a frequency array F that keeps track of how many times a bit was changed essentially. So, initially of course, when you start of F is 0s for every bit. But, at some point let us say that F is F 1 number of bits number of times that bit 1 was changed F 2 is the number of times that bit 2 was changed and so on.

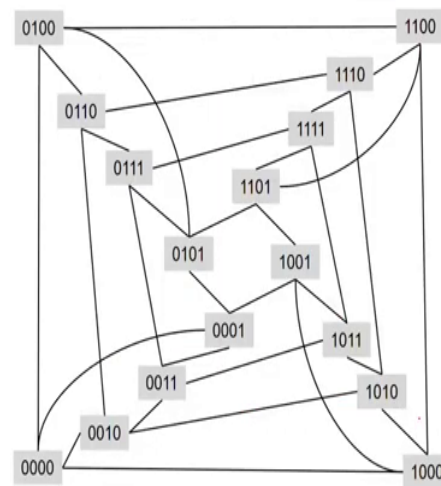
And so for so for example, if F 1 was 100 and if F 3 was let us say 2 100 and 12 and if F 5 was only 5 which means you have changed F 5 very few number of times. We would like to encourage this fifth bit to be 5th bit to be changed more often essentially. How do we do that?

We do this by penalizing the move or penalizing every move each move by a factor which is proportional to the frequency. So if node i is generated by changing the i th bit then you penalize the value of that node which we are calling an evaluate eval value. So, that is a value of this node i . By subtracting the corresponding frequency count or something proportional to the frequency count from that value and this subtraction is for a maximization problem.

If you are doing a minimization you might add something to it because in minimization large values are bad essentially. So, in this way we are trying to kind of push the search into areas which has not been explored often enough. And when we moved to stochastic algorithms in particular when you move to this algorithm called simulated annealing we will see that we will try to do this in a different fashion is that we would try to add an element of exploration to the heuristic based search that we have been doing so far.

(Refer Slide Time: 22:44)

The solution space for a 4 variable SAT problem



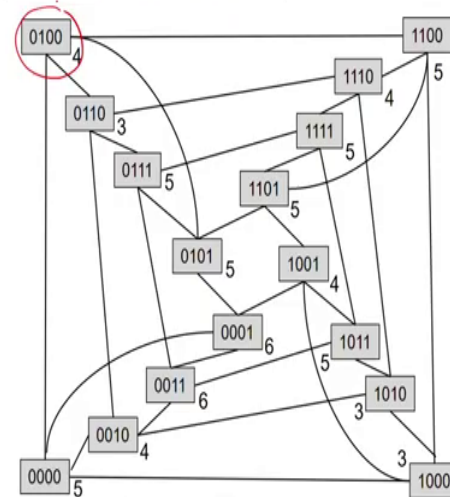
So, let us look at an illustration of local search and in particular tabu search for SAT problems. So, here we have drawn the complete state space for a 4 variable SAT problems.

So, remember that for a 4 variable SAT problem the number of states is 2 raise to 4 which is 16 and we have drawn the 16 states here and since it is a 4 variable problem each node has 4 bits that you can change. So, each node has 4 neighbours as you can see from this diagram and we are just arranged this candidates thanks to Baskaran in a nice fashion which look visually appealing essentially.

So, this is a complete state space for sorry complete solution space for a 4 variable SAT problem. In practice, the search algorithm will not first generate the space and then explore it as we know our search algorithms they generate the spaces on the fly, but this is just for the sake of explanation we have drawn the complete state here. If the number of variables for anything larger than 4, then we would probably not be able to draw such a figure.

(Refer Slide Time: 23:57)

A formula with 6 clauses



$$F = (a \vee \neg b) \wedge (\neg a \vee b \vee c) \wedge (\neg c \vee d) \wedge (d) \wedge (\neg a \vee b \vee d) \wedge (\neg a \vee \neg d)$$

$h = \text{number of satisfied clauses}$

Artificial Intelligence: Search Methods for Problem Solving

Deepak Khemani, IIT Madras

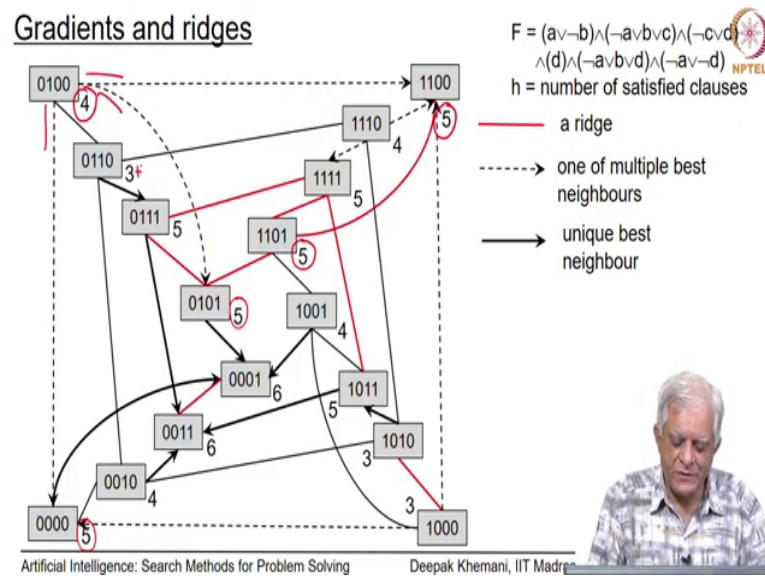


So, let us now impose a problem onto the space. The problem is imposed by choosing a formula which as you know is a boolean formula F we call this formula F and it is given as set of 6 clauses. The first clause is a or not b , the second clause is not a or b or c and so on. So, we have 6 clauses and as we have discussed earlier the heuristic value is going to be the number of satisfied clauses. So, obviously, in the case of a solution all the 6 clauses must be satisfied.

So, the value of a goal node is going to be h equal to 6 and this is what we would try to achieve and you can also observe that this is a maximization problem. Because other nodes will have only a smaller number of clauses and we want to move towards them. What this diagram shows is the value heuristic value for each node.

So, the top most left corner you can see has a value 4 which means 4 of those 6 clauses have been satisfied for the assignment a is equal to 0, b is equal to 1, c is equal to 0 and d is equal to 0. So, that is this node here and likewise for every other node you can see that the heuristic value has been depicted here.

(Refer Slide Time: 25:29)



What this diagram shows is what is the nature of the terrain. We have said that the heuristic function is what defines the terrain and in this diagram we have tried to demarcate 3 different kinds of edges. The first one is shown in red which is what we call as a ridge. So, for example, this value 5 is connected to an equal value 5. So, it is like a ridge in some landscape which means that a hill climbing algorithm would not make that move essentially.

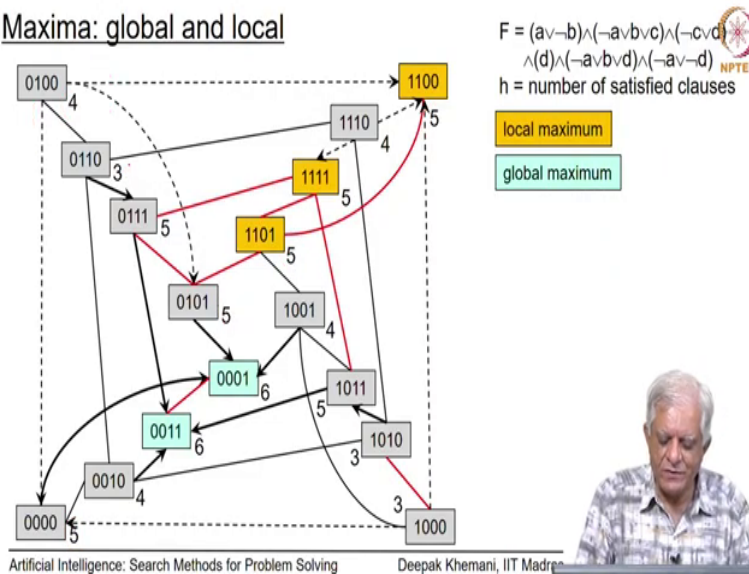
If you are looking at a node with heuristic value 5 or evaluation value 5 and if the neighbour is 5 since the neighbour is not better than the current node hill climbing would not proceed. But,

what we will show now shortly is that a tabu search would indeed work with this algorithm. The second kind of edge is shown in this dotted arrow and what this shows is that there are multiple nodes multiple neighbours which are best and all of them are equally good.

So, for example, for this node with value 4 there are 3 nodes whose value is 5 and who are neighbours of this particular node. And so we have drawn 3 dashed edges coming from there. So, 1, 2 and 3 and there is one node which is worse which of course, the algorithm will not moved with. So, a hill climbing algorithm starting with this node 0 1 0 0 could make any one of these 3 moves which all take it to an heuristic value of node 5. The third case that we have identified is a thick arrow and this thick arrow shows that there is exactly one best move and therefore, the hill climbing would make that particular move essentially. So, we are just trying to get a feel of the terrain of what it looks like.

(Refer Slide Time: 27:50)

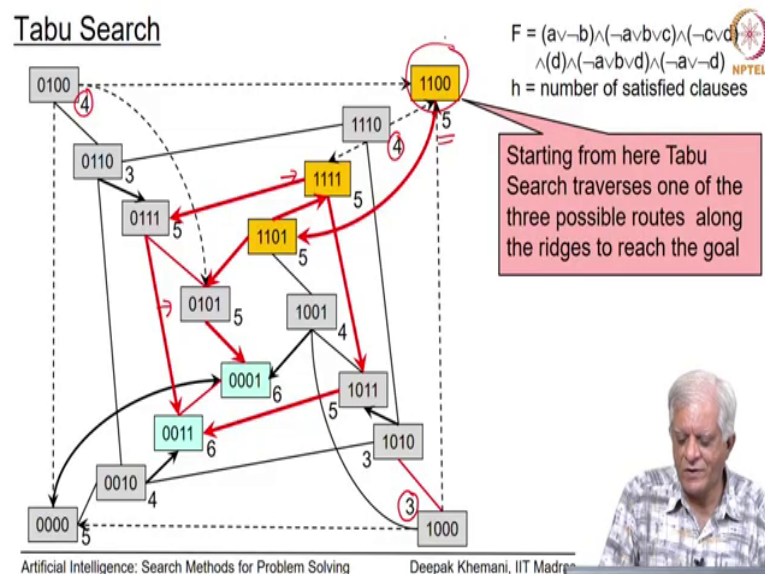
Maxima: global and local



So, another aspect of the terrain is what are the values? So, if you now look at this diagram in which we have colored some nodes. The orange nodes are local maxima you can see that there are 3 of them all of them with value 5 and that is because all their neighbours are either equal to 5 or less than 5 essentially. But, we also have 2 green nodes which are the global maxima whose value is 6 which is a solution that we are looking for this particular SAT problem.

So, we can by looking at this diagram we can say that this particular formula would be satisfied if the last bit was 1 and it would also be satisfied whether the second last bit was 0 whether the second last bit was 1 that does not matter. The first 2 bits have to be 0, the last bit has to be 1, the 3rd bit can be either 0 or 1 and in these cases it is a global maxima. So, the goal of our search algorithm would be to reach this global maximum.

(Refer Slide Time: 28:52)



So, supposingly we want to start with this place which is a local maximum which is the node 1 1 0 0. You can see that it is a local maximum which means that it has neighbours whose values are either equal to or less than this value. So, one neighbour has value 4 which is less than 5. So, this is 5 that is our current node. Another neighbour also has a value 4 which is less than 5 and a third neighbour has a value 3 which is also less than 5.

There is a fourth neighbour which we had not drawn, but whose value is 5 and in fact, that is what is the path that tabu search would take. It will go from this node to this it will just traverse along the ridge and go to another node with a value 5 in the first cycle. So, in the first move.

So, in the first move this would make this move, then this node has more than one neighbours connected by ridges and it could move to either one of them. One of them is another local maxima which is 1 1 1 1 and a second one is not a maximum, but it is 0 1 0 1. So, it could move to any one of them from these 2 moves which is this one and this one it has further 3 choices.

So, it can either move to the goal node which is 0 0 0 1 or it can move to along 2 other ridges 2 nodes which are also of value 5, but which are not goal nodes. From there again in the last move you could move to the goal node. So, you can see that starting with this node 1 1 0 0 hill climbing would have got stuck on this node, but tabu search would have found a path to the goal. In fact, it would have found one of the 3 possible paths to the goal and succeeded essentially. So, this kind of gives you an idea of this power that tabu search has of moving off the local maxima. In this case, the start node is a local maxima and eventually finding a path to the goal node.

So, in this particular example it only moved along edges to node which had value 5, but you can think of another example where it might move to a lower node as well essentially.

(Refer Slide Time: 31:39)



Stochastic Local Search Methods



Artificial Intelligence: Search Methods for Problem Solving

Deepak Khemani, IIT Madras

So, we will stop here with deterministic algorithms and in the next session we will move to stochastic local search methods which are kind of you know the class of algorithms also called as randomized algorithms. So, see you in the next session.