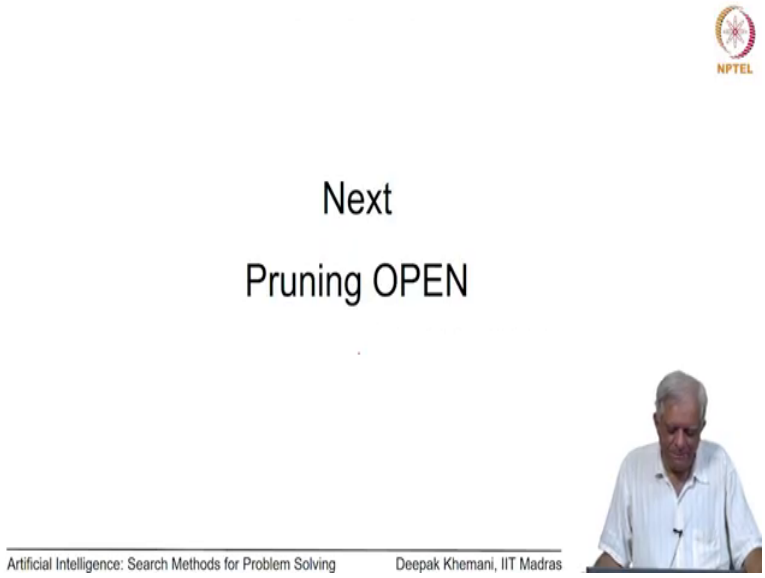
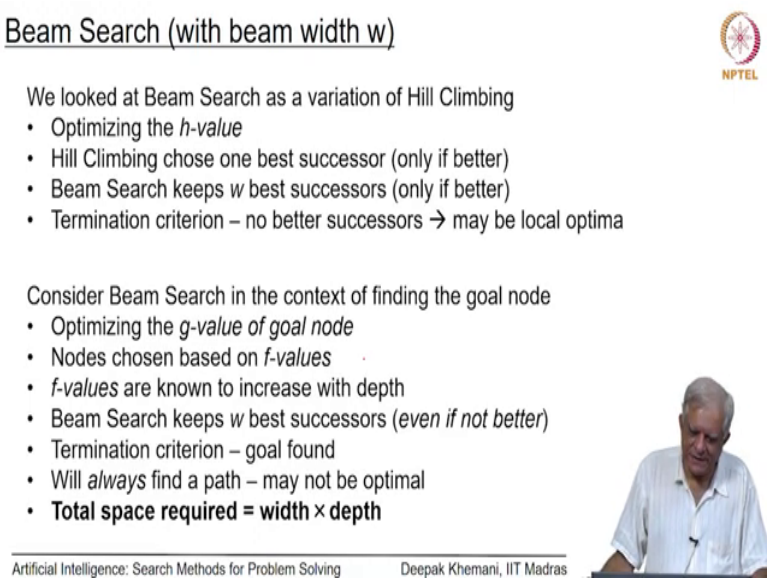
Artificial Intelligence: Search Methods for Problem Solving
Prof. Deepak Khemani
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Chapter – 05
A First Course in Artificial Intelligence
Lecture – 48
A*: Pruning Closed and Open Breadth First Heuristic Search

(Refer Slide Time: 00:16)



(Refer Slide Time: 00:20)



Now, we are looking at other ways of deleting open node or pruning the open node. We already have some idea of doing this when we looked at the algorithm called beam search. Let us assume that, we are doing an algorithm called beam search with width d with width w.

And we looked at beam search in the context of hill climbing; because hill climbing kept only one copy at each node. I kept only one best successor; but in beam search we said that why keep only one, you can keep w successors, because you know that much memory would be required and it would still be a constant space algorithm.

The termination criteria for hill climbing and beam search when we studied in that context was that, if you do not have a better neighbor; then you terminate, which meant that you could often end up at a local optima.

This process happens, because it moves to a successor; both the algorithms hill climbing and beam search move to successors only if they are better essentially, and that is why we call it the steepest gradient ascent or descent as the case may be algorithms. And it moved along the slope till they were slope became zero essentially.

But now, let us look at how we can adapt this idea of beam search? So, what is the basic idea of beam search? Is that at every level as you go deeper into the search tree, you keep only a fixed number of nodes and that is called the beam width and let w be the beam width in this case essentially.

So, when you consider a beam search in the context of finding the goal node, that is our goal in the A star algorithm, find a path to the goal node. In fact, in A star we interested in finding an optimal path to the goal node. And in that context, we can see that, what we are optimizing here is the cost of the path that we found to the goal node and that cost is the g value of the goal node essentially.

In earlier, we were trying to optimize the heuristic value; because we said that for example, in a search, in path finding kind of a situation, the goal will have heuristic value zero and therefore, we try to see where is a node which has the lowest heuristic value. Now, our goal is different, our task is different; in the sense that we want to find an optimal cost paths to the goal node and that is the g value of the goal node.

The algorithm itself does not use g values, it algorithm uses a combination of g and h values and we call that as a f values of the goal node. Now, we have studied that when especially in the case of monotone condition and even otherwise; f values tend to increase as you reach closer to the goal.

And the reason for that was the contribution of h decreases as you go closer to the goal, and the contribution of g increases. And h is the one which was under estimating the cost to the goal and therefore, as you go closer to the goals, g plus h together as a whole increases.

And this is specially so in the case, where this heuristic function satisfies the monopoly property and we showed that; that as we go towards a goal, the f values increase. So, clearly we cannot use a criteria that we use earlier which is that, you move to a successor only if it is better; our criteria is not to find the path to goal node.

So, we will say that, move to the successor which is the best successor, irrespective of whether it is better or not. So, this is a little bit like Tabu search if you remember; in Tabu search we were not so fixated with slope, we said that we can move off a local maxima and move to the next best node and so on. But there is the concern so different that you do not want to get back to that local maxima; so we had this Tabu list and so on.

Here we are saying that, we will move off the local maxima; in fact it is not even local maxima, we will move to worse nodes. But, because of the fact that g values will keep increasing; we will not go into cycles going back and close because every time we go into a cycle going back, the cost of the path found, which is a g value will increase essentially.

So, the first thing we have to make a change is that, it does not matter if the f value of the neighbors is higher; we expect it to be higher in infact and just move to the best successors, best f value successors. So, the beam search in this context will keep the w best successors; remember w is the beam width, even if they are not better. And in fact as I keep saying, we expect them not to be better.

The termination criteria is different and that is you will terminate only when the goal is found essentially. And we can argue, we will not do so formally here; but you can go back to the argument that we made during A star when we said that, even for infinite graphs, the algorithm will terminate with the paths to the goal if there exists one.

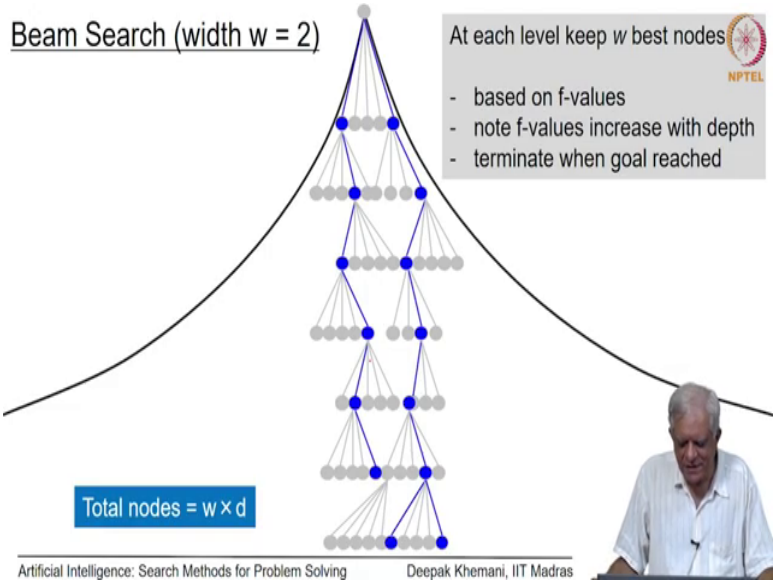
And we had used the argument, if you remember that every edge cost is greater than some positive number epsilon; which means that, if you go along paths which are not leading to the goal, the g value will keep increasing and eventually they will become so high that some paths to the goal will be found.

So, beam search which moves to successors which are in fact higher f value will still find a path to the goal; but it may not be the optimal path. The important thing about beam search for from for us which is from the space perspective is that, it requires less space; the total space required, which is open plus plus close is width into depth.

Width is the number of nodes at each layer that you are keeping, and depth is the number of layers that you are exploring. So, the space requirement of beam search is linear in nature; the further you go, you only need to have a linearly increasing space requirement.

Or in other words if you remember it means that, at every level you go deeper; you add a certain number of nodes to your space requirement essentially, instead of multiplying which is the case for exponential growth. So, linear growth means, you just add some w more nodes at every level. So, that is the main, main reason why we are again looking at beam search, just to save on space.

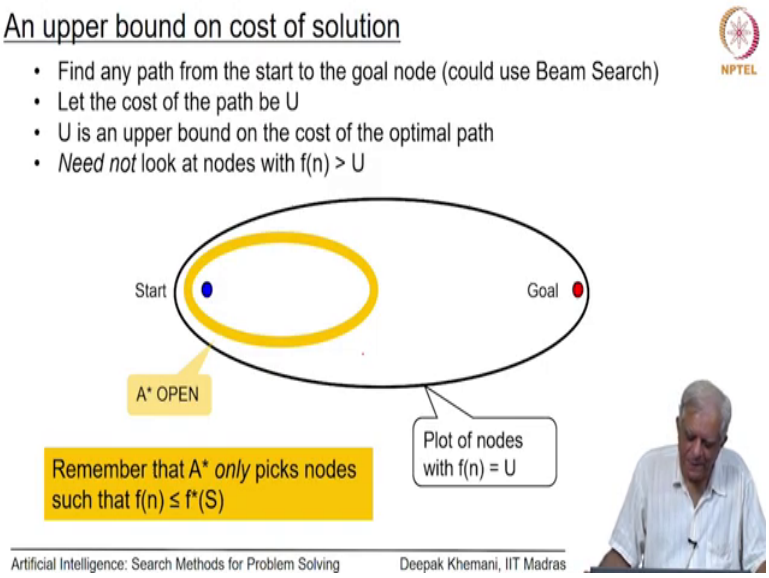
(Refer Slide Time: 07:14)



So, this is our beam search looks like, if the beam width is 2; then at every level we keep two nodes, which are the best nodes, which are shown here in blue color essentially. So, as you can see, the rest of the nodes which are in gray, we do not need to keep them; and we can, we will in fact delete them essentially.

So, we will only keep two nodes at every level and based on how much deeper you are, that many times width in this case two is a space requirement. But remember that f values keep increasing and we will terminate, not when we reach some kind of a zero gradient; but when we have reached a goal state. And we have argued just now that, we will eventually reach a goal state and that is because, the g value will otherwise keeps increasing too much.

(Refer Slide Time: 08:09)



Now, given this beam search, we can now find an upper bound on the cost of the solution. We already know a lower bound on the cost of the solution; in fact the lower bound is the optimal value essentially, that the solution cannot be less than the optimal value. Now, we are talking about an upper bound and this is going to be for the purpose of controlling our search to not go into regions which are going to be too expensive.

So, how can we find, find an upper bound? Simply find any path from start to the goal and you could use a beam search algorithm. Fine it may not return optimal path, but it will return a path. So, once you know that there is a path to the goal node, let the cost of the path be U.

And you can see that now U is an upper bound on the cost of the optimal path; the optimal path is going to be less than or equal to U. But more importantly any node which has an f value greater than U, can be ignored.

And again the reason for that is that, f values underestimates of the actual cost. And if you know the, if you know an upper bound and if the f value which is an underestimate is more than the upper bound; then definitely you do not want to look at that f node that, that node.

So, need not look at nodes with f value greater than U; because f values are themselves as we said underestimates. So, this is what the picture looks like; if you have a start node and then goal node, and you use the beam search algorithm to find one path to the goal and use the cost of that path as a kind of a boundary, as a kind of a lakshman rekha, beyond which you will not allow search to venture.

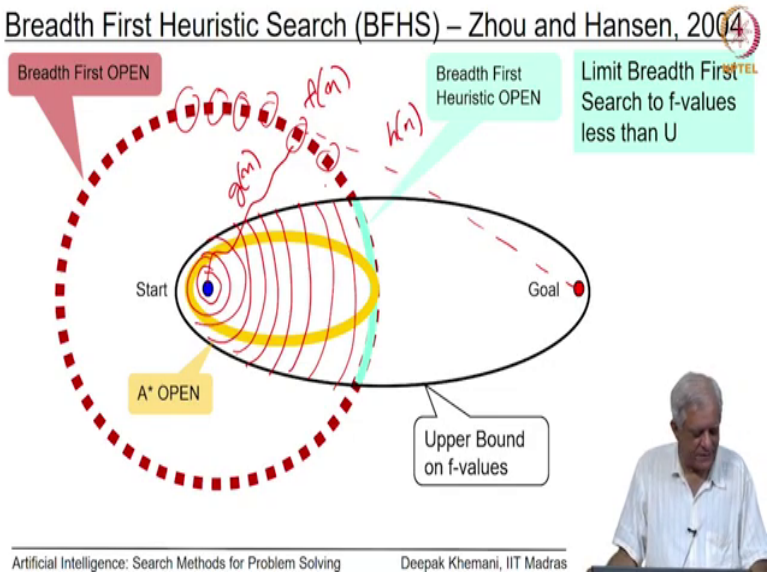
Now, the shape of this I have drawn as an oval and that is simply to illustrate the idea; but it will really depend upon the quality of your heuristic function. If the heuristic function is very good, this would be a long elongated oval; if the heuristic function is poor, then it will be more like a circle essentially, we will see that shortly.

But in general, if you plot all the f values, whose f values equal to U or let us say greater than U also, then we will get a boundary like this. And now we know that any search algorithm that we use must stay within this boundary; because outside that boundary, there is going to be more expensive paths. And we already know that there is at least one path which is a cost of U essentially.

So, this will serve a very strong influence on controlling the algorithms that we are going to see now, from preventing them from going off in a wrong direction essentially. Now, the open list of A star would look typically look like this, at some stage of the search algorithm and you must remember that, A star only picks nodes whose f value is less than or equal to the optimal cost path.

And the optimal cost, we are just argued is going to be less than U essentially. So, the open list of A star would typically be somewhere within this boundary that we are talking about; but as some researchers, in fact Zhou and Hansen themselves found, it can empirically be still formidable.

(Refer Slide Time: 12:06)



And they came up with an algorithm which they called as breadth first heuristic search; the title sounds a little bit like a contradiction in terms, because on the one hand we are saying it is a breadth first search and now other hand we are saying it is a heuristic search.

But it is quite interesting to see and let us see what they meant by this. So, let us get back to our search problem and see that if you are doing breadth first search; then what would your

open look like? Now, breadth first search has no sense of direction at all; so it would explore the paths in all the directions.

So, kind of schematically we can say that the search open frontier will be like a circle centered on the start node. So, in all directions from the start node, it would consider all nodes equally good, and breadth first would be like open. But just now we argued that, we would like to not allow search to go beyond a certain boundary and we will now limit this breadth first search; the search will be breadth first.

But we will now limit to f values which are less than U, that is used the value we found by some algorithm maybe a beam search algorithm. And so, this was the boundary within which we were confined to stay and there is no point looking at all these nodes which are outside these f values; because if you consider the path to any one of them and the path from there to the g. So, this is some g of n and this is h of n.

And the sum is f of n and this f of n is more than U; so no point exploring such nodes. So, what breadth first heuristic search does is that, it is searches in a breadth first fashion; but it stays within the boundaries given to us by the upper bound which is given to us by some paths that we have found.

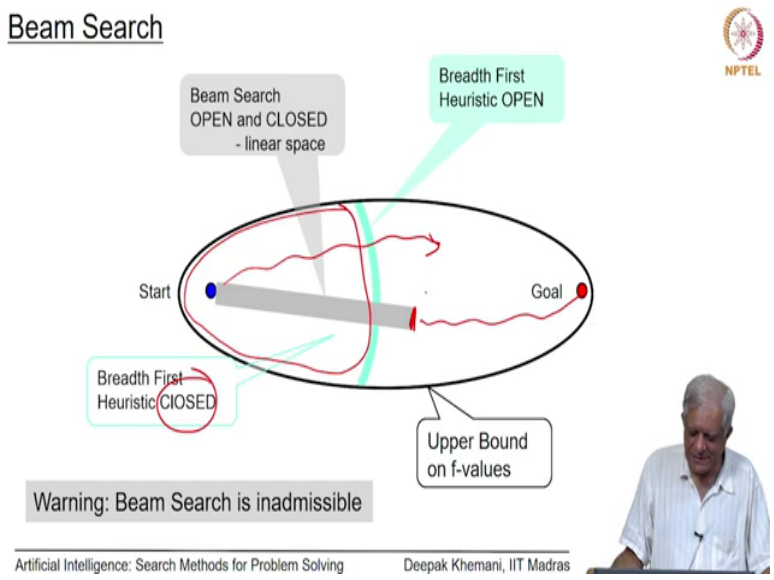
So, what is going to be the behavior of breadth first search? It will first circle around the start, then it will circle around one level deeper; then one level deeper, then one level deeper. So, it will keep doing like this, but never venturing beyond the boundary. So, this is how the search will proceed.

And what they empirically found was that, the open list maintained by breadth first search, breadth first heuristic search; heuristic because it is now using the upper bound U, which has been discovered heuristically.

They found that, empirically the open list of breadth first search is much smaller than the open list of A star, which is kind of suggested by this diagram; but the diagrams only

schematic. So, they should not be taken as proof of any kind; but just as suggestions that, it may be the case, especially if the heuristic function is very good essentially.

(Refer Slide Time: 15:15)



Now, let us get back to beam search. What does beam search do? Given the same upper bound and given that breadth first heuristic has an open which looks like this. So, what is the space requirement of breadth first heuristic search? It is open plus closed. What is closed? Close is everything on the left hand side of the open node within the boundary. So, I am not shaded those nodes here; but you can imagine that all these nodes which fall within this region, they are in the closed list essentially.

So, this is a total requirement of breadth first heuristic search. What about beam search? Beam search is a simply keeps w nodes at every level. And so, open and close together as we have argued, close linearly. So, breadth first heuristic search it will again still depend upon

the how good the heuristic function is; the better the function, the narrower will be the boundary. But in general we have kept saying that, heuristic functions are never perfect essentially.

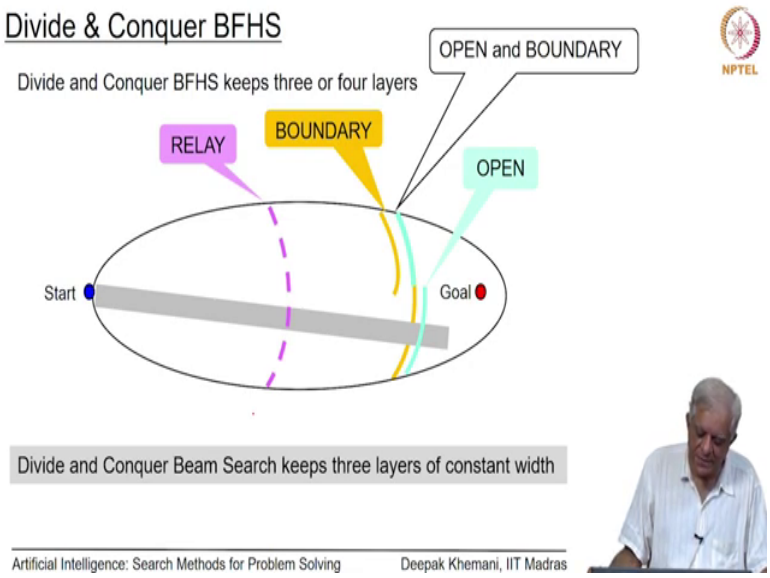
Beam search as we have said, our main attraction towards beam search is because of the fact that, it requires linear space together for open and closed. So, open is only at the front edge of this and the rest of this shaded region is a closed essentially.

And we have seen that it is close linearly; w into d, where w is the beam width. But of course it comes with a warning that, beam search is not admissible; it will not give you the optimal path, it will it will definitely give you a path, but not necessarily an optimal path.

So, if you want an optimal path, then you better go to something like A star algorithm and the characteristic of A star like algorithm is that they never preclude a possible solution. So, this is something that we started over saying that, arranges search space, which does not preclude any solution.

So, in this case, because the solution may have lied around this path; beam search will not find it, because we have said that we will terminate as soon as we, find the path to the goal. And that path could be somewhere along this line, which may not be the optimal path. So, how can we now adapt or adopt some methods to adapt the beam search into admissible algorithm; which means that, it will find the optimal path for us?

(Refer Slide Time: 18:02)



Before we do that, we can combine the methods that we studied for pruning closed. So, remember that all these algorithms were this idea of keeping a layer and a boundary layer and an open layer, came from pruning closed. The idea of breadth first heuristic search came from pruning open.

If we combine both the aspects; then we do not have to keep the entire closed list for breadth first heuristic search, we keep only a boundary layer and the relay layer, like we did for the earlier algorithms that we saw, but of course at the expense of having to reconstruct the path again and again.

We could do the same thing with divide and conquer beam search, it will only keep three layers of constant width; one layer for open, one layer for the boundary, and one layer for

relay nodes. So, this divide and conquer versions of breadth first heuristic search and beam search, essentially try to prune both the open and the closed list essentially.

But let us get back to first trying to see, whether somehow we can make beam search a little bit more explorative in nature, so that it will not ignore any paths to the; it will not ignore any paths, which means that it will find the optimal paths essentially.