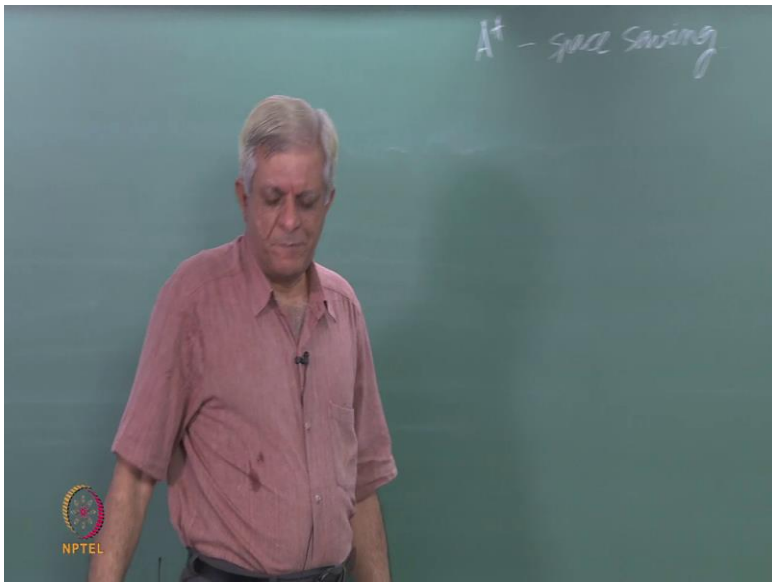
Artificial Intelligence

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

Lecture - 22
Recursive Best First Search, Sequence Allignment

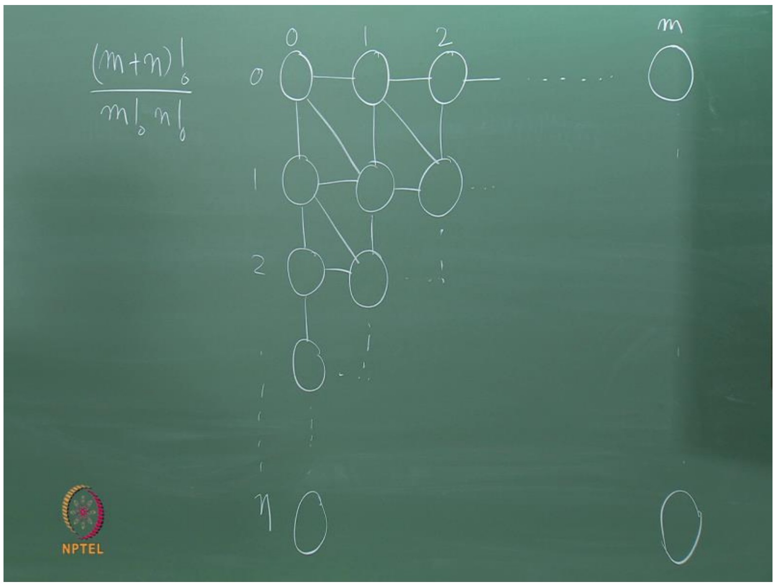
We have been looking at algorithm A star, which is a very well known algorithm for finding optimal solutions in a state space.

(Refer Slide Time: 00:27)



And at this movement, we are trying to look at variations of A star which are space savings. So, before I start the algorithms, let means ask you one small or status small problem.

(Refer Slide Time: 00:59)



Let say you are in a ((Refer Time: 00:53)) like city and there are a set of so, it is a grade like this. And you have to come to this last city here. Let us say, this cities are numbered 0, 1, 2 up to m and 0, 1, 2 up to n. So, it is on m by n grade and by this I mean m adjusts in this direction and n adjust in this direction. There are m plus 1 cities in horizontally and n plus 1 cities vertically essentially. It is a complete grid, I am not drawn the complete grid. The question I want to ask is how many paths are there, if you want to go from this city 0, 0 to the city m, n there.

And we are assuming that you can only travel either to the right or down essentially. So, it is not, it is a directed graph, you can only move either to the right or down. But, the question is how many paths are there, m and n you have to use two parameters m and n essentially, whether I m adjust here and n adjust in this direction. So, if you if there only one city, there is only one path. If there are four cities then you can see there are two paths. As the number of cities increased, the number of paths increases quite dramatically. Thus, anyone know of the korf how many.

Student: x plus y c x.

x plus y…

Student: C x, I mean total number of either we can go in as an if one direction is fix than the other direction could be fixed, suppose from all the paths you choose after fixing one direction on to the limited numbers left.

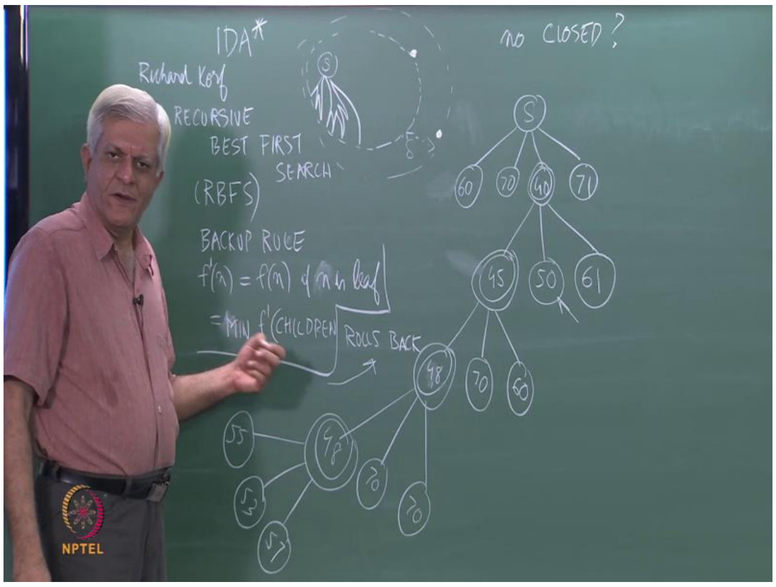
Well, you can think of this as a sequence of right moves, we have to always make m right moves and you have to put in n down moves, soumiya.

Student: n plus n factorial ((Refer Time: 03:34)) divided by n factorial.

Divided by n factorial, that is a correct answer, you can think of it has saying that you have to make m right moves and somewhere along those m right moves, you have to put those n down moves and the number of ways we can do that is given by this essentially. So, I will come back to this problem; let me increase the complexity of this problem, what if I allowed diagonal moves as well. Now, that is a little bit harder, it does not have a nice closed form solution, it is the summation of many things, but I would like it to think about that.

So, you can think of this new problem as follows that one diagonal move replaces one horizontal move and one vertical move. And out of the m plus n horizontal and vertical moves that you are making, whichever is smaller m or n, you can replace it with that many diagonal moves optionally, you do not have to, but you can replace it with that many diagonals moves. So, that many horizontal and vertical moves become less and diagonal moves will become more essentially. So, this a little bit more difficult to analyze, but it is a good exercise.

(Refer Slide Time: 05:18)



So, let us first continue with our variations that we were looking at. In the last class we saw I D A star and if you recall, what I D A star does is that, essentially from the start node, it creates a boundary, this is the goal. And this boundary is the heuristic value of the start node or in other words how far it things, the algorithm thinks the goal is from the start node. And it draws at boundary and does that depth first search on this. It rate a ((Refer Time: 05:54)) comes on the fact that after it is fail to find a goal here. It would have some nodes open just around the boundary, which it has not expanded.

It looks at those nodes and fix the minimum f value from there, and extends a boundary by that much amount essentially. And it keeps doing that, till it has found the goal node essentially. So, I think we are started discussing the properties of this algorithm, but let us recap. So, maybe I will ask you to tell me, what do you think of the I D A star algorithm. It was formed by Korf, Richard Korf 1980’s something, I do not remember exact date. So, what is good about this algorithm and what is not good about this algorithm. So, the first question we want to ask is does it survive the substitute for A star, which means does it guarantee and optimal path or not. So, what is the argument you would give?

Student: ((Refer Time: 07:18))

Why, how can you say the ((Refer Time: 07:21))

Student: If I do not the optimal path at a certain value of that then what I am doing is I am finding the node with the value of f just higher than that. So, I would not miss out anymore. So, the movement I the value of f, which is the ((Refer Time: 07:37)) goal I will find ((Refer Time: 07:39)).

So, that is one criteria it satisfies for us, it is that it finds the optimal path, why is it better than A star, why would you prefer it to A star or when would be prefer it to A star.

Student: Open lays does not go too large.

Open lays does not go too large in fact, it goes linearly only and especially, if you are looking at problems, in which thus search space is very huge. You can imagine if you are generating something like hundred thousand nodes a second in the search space. Then how quickly well I am will get fill up with, you remember that each state, each node is a representation of a state which means that, whatever you have described about the state would be there. And then you have applied a move gen function and you generate a new state and you put all those things into some kind of a data structure essentially.

So, the advantages of I D A star is that it needs less space and this is the theme that we are going to follow today, how can we look at space saving algorithms, but what is a disadvantage that you can think of.

Student: ((Refer Time: 08:57)) time complexities.

So, what do you mean by saying better pass may be left unexplored. We have already argued and he said that, we guarantee that it will give you the optimal solution. So, that is what we want essentially. Time complexity is like it was in the case of D F I D time complexity is an issue, but unlike D F I D. In D F I D we assume that the cost was uniform of each edge which meant that, as you get went deeper, the number of nodes in the next layer was increasing exponentially or it was multiplied by the branching factor b.

And therefore, every time we increase the depths by one, you encountered many more new nodes in fact, more you nodes then the old nodes that you had seen so far. Unfortunately for ideas are that is not the case, because now we have edge cause involve and as we discuss, we increase the bound only to the next lowest f value, which means that it may do a significantly large number of searches before it actually converges to this, which is why we observed in the last, towards the end of the last class that instead of incrementing it to the next lowest step value. You increment it by some amount, we determined amount delta, which is a loss of optimality or willing to bare essentially.

So, if you increase further by a value delta, this is delta. Then the next time it searches, it may find a node somewhere here, which is a goal node. So, I have drawn only one goal node, but in practice, search problems may have more than one goal node essentially. So, for example, if you are looking for a Chinese restaurant in a city, they may be many Chinese restaurants, if you are looking for a particular film, which is showing in some theater, you may have more than one solution essentially. So, depends on what your search algorithm finds.

So, it might find a goal node here, as oppose to a goal node here, this is close to the previous bound that we had, but this is far there from the previous one. So, it has an error of our delta. So, if you are willing to tolerate an error of delta, then you can increment this delta and you can control how much you have willing to tolerate by essentially. So, that is one problem with ideas are that it may do too many iterations essentially. Another problem with I d A star is that, it is sort of does not cater to our ascetic sense if you want to call it, in the sense that, it does not have a sense of direction.

We started search with line search algorithm and then say, then we said we are introducing heuristic functions to guide search towards the goal. The only role the heuristic function is playing is in defining this boundary here. Of course, the boundary as you can see is ((Refer Time: 12:05)) sided, it is towards the goal by that is about all it does essentially. So, the next algorithm that we want to look at, which is also by Richard Korf is called recursive best first search, popularly known as RBFS.

So, R B F S is a little different from I D A star and no doubt korf devises algorithm, because he saw the drawbacks of I D A star, which is that, it was doing too many iterations. In particular, if you do not maintain a close list and you just let, D F S in some sense run wild is that was good idea or bad idea to do. Let us say, D F I D or I D S are without maintaining a close list. With it get, the question is one of the things at closed, thus for us is there it is stops as from going to an infinite loop essentially.

So, if you have for example, search after this is, this I told you was a directed graph, but if this one not a directed graph, then you could have gone into a loop keep going in a loop like this essentially. And close allows us to avoid such a possibility, but given the fact that we have working with bounds on the distance from the source that you have willing to go, can I work without closed? In other words, if you go back to the D F I D algorithm, which is basically a simpler version of I D A star that it does not have edge cause will D F I D work, if I do not, if I implement the D F S without a close list.

So, I will leave that the small topic experiments for you to do. Let us forget this algorithm RBFS. So, what R B F S does is that, it maintains a linear amount of memory exactly like I D A star, but it does not. So, what I D A star is doing, it is doing blind search, wherever the goal is it will go of in one direction, back track, try something else, back track, try something else, back track, try something else and so on. So, it is will basically a depth first traversal of the space with no sense of direction. Now, let us try to stimulate, what R B F S could do.

So, let us say we start with this node start, and let us say we have these four children, and their heuristic values for the sake of argument, let us say, this is 40 and 70 and 71. So, initially recursive best first search behaves like best first search, that it fix the one with the lowest f value and expands that, which means in this example this know. But, what it also does is that, it keeps a pointer to the next best node that is in the open list. So, in this example, this is an express node and then it does the search essentially.

So, this is second best node and that is it node ((Refer Time: 16:06)). So, let us say it generates this and if you remember the monotone criteria are consistency conditions that we said, that they in generally you expect the heuristic values to become more accurate.

As you go closer to the goal and a consequence of monotone criteria was that f values increase as you go forward, as you go towards the goal essentially. So, he would generally expect f values to increase in a search space, which is consistent. So, let us say, this is become 45, this becomes 50 and this become 61 for arguments sake.

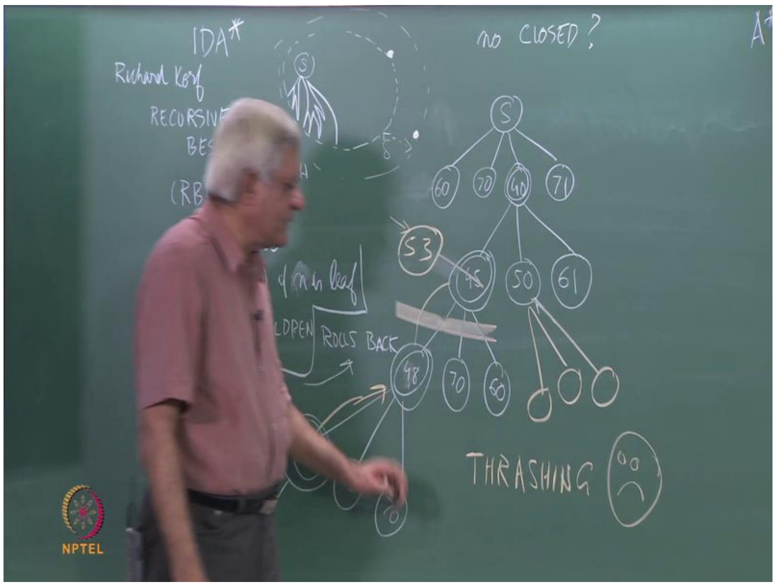
So, what immediately R B F S will do is, it first removes this pointer from here, and put it to this. Because, now that is a, to this, this is the best node and this is the next best node in the open list, and it keeps a pointer to the next best nodes essentially. Then it goes like this and let us says this is what is happening, let say this becomes 48 and this becomes 70 just for arguments 60. Then, it expands this one, remains 48 let us say and just for argument sake, let say these are all 70 and it expands this. So, it is diving into the search space, using the heuristic function as a guiding whose.

Let us say some point, these nodes start looking and little bit worst than that node. So, this becomes 55, this becomes 53 and this becomes 57 or something like that. So, so far it is behavior and the best for search behavior is identical. But, at this point well, it has to what best first search would have done is simply, it would have pick this node as the next one from the open list and started exploring the tree from there essentially, what recursive best first search, which is trying to save one space, thus is that it rolls back this search all the way. So, that it can move to the next siblings essentially.

So, it is basically deletes all these nodes from open, close whatever it is maintaining. So, it rolls back you know, there is to say about the Indian government at one point, it is a roll back government. So, they would increase the petrol prices and after two days roll it back or something, little bit like that. But, it has a rule which it follows and which is called as a backup rule, which determines the f value of a node, f of n is equal to all let us call it some other value. Let us call it f prime value is equal to f of n, if n is a leaf which means n is on open is equal to max no min f prime, I just use this short form.

So, either it is a f value or it is a minimum of the f values of it is children essentially. So, what really it does is that it maintains this f prime value. So, for all these nodes 70, 70, 70, 60, 57, 53, 55, the f prime value is the same as the f value, because it is a leaf node. But, when I it rolls back, it applies back up rule to back up the values. So, for example, from these three nodes it will back up the value 53 here.

(Refer Slide Time: 20:48)



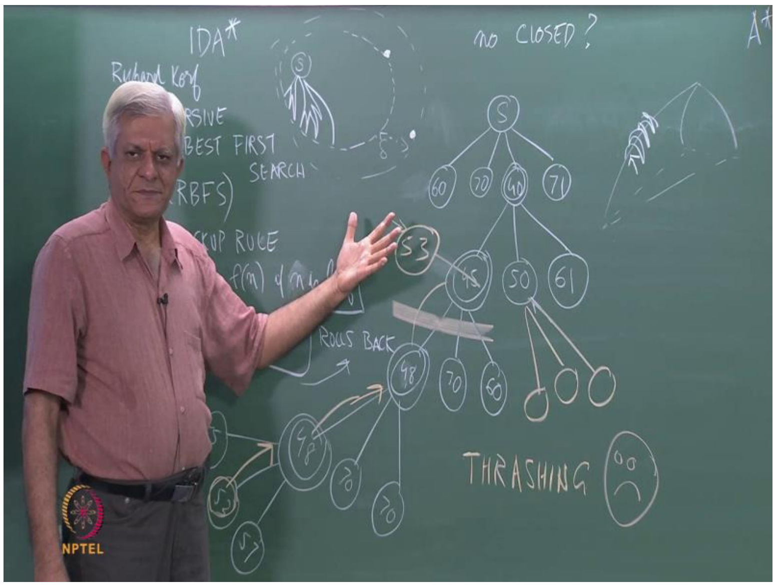
So, let we use a different chalk to illustrate that. So, it backs up this value 53 to this node. So, now this becomes 53 and there it deletes this essentially everything. So, this is 53, 70, 70, so again it backs up 53 here; this is 53, 70, 60. So, it backs up this thing here, changes this to 53, which is the backed up value and then to seats along this direction. So, you can see that, once it has devises value to 53, if you just look at the snap shot of the search at that point, all these nodes are not there, all these has been deleted, only that much of the tree remains. And this is on open, this is put back on to open with a new value of 53, this is 50, this is 61, this is 70.

So, it basically naturally goes off on that direction. So, you can see that at any given point, recursive best first search maintains only one paths down the search tree, which means it is space requirement is going to be linear, because that is what the depth first search also does. And yet, it mimics the behavior of A star algorithm, in the sense that it is best first, it always goes down the best looking path. It just that on the way, it has revise the value of this node, the heuristic value has measure by the heuristic function was 45, but after is done this search, it realizes that it is not 45, it is 53.

So, it leaves it to the open value of 53 and goes down this path essentially. So, it will generate these children now, and depending on whether they are. So, when this is 50, this will become the next best node essentially. So, if I going down this path, there is no node better than 53 in this path, it will roll back from here, and go down this path again. So, you can sense a similar problem as in that it may do this many time, it may go down this path, it may go down this path, it may come back here, eventually it may finish this and it may go down this path.

And then maybe, this will become worst and then it will go down this path. So, R B F S is a danger that it opposes it, depicts the behavior of what we call is thrashing, that is a danger with R B F S, you read that it may thrash something like this. So, I do not know whether we discuss thrashing, when we were looking at hill climbing, but if you imagine a hill, which is like a ridge.

(Refer Slide Time: 23:38)



So, which looks a bit like this cross section if you can visualize this, like this? So, it is like a ridge essentially, which is slowly increasing in one direction. Then hill climbing would have a tendency to go in this direction which is, because that is a steepest gradient direction essentially. And if you are search problems is the granularity of your move gen function is search, that once it makes a step in this direction, it may over shoot at some point then it make, it may come back that may go like. So, hill climbing can also behave a similar fashion behavior essentially.

So, this is the one problem with search algorithms, which are local in some sense essentially. This is local in the sense, it is always going down one path, but it is not completely local in the sense are it is keeping updated values, backed up values, for known essentially. So, R B F S is came around 1990 also was an improvement on I D A star, because it had a sense of direction, but it has this problem there, it could do ((Refer Time: 24:46)). And actually we have students, we have implemented these algorithms for the course here, I have observed this behavior that you know these algorithms just keep spends, what they say as infinite amount of time, you will just touching between a few loads essentially.

So, what you would be really interested in is space saving algorithms, which behave more like A star in terms of the nodes that they are picking for expansion as well. Whereas, these two algorithms behave like A star in terms of the solutions they produce, the way they pick nodes is different. And consequently both this algorithms have a of course, much larger time complexity in A star, because they will explore the same space again and again many times essentially.

Now, so let me now ask a question has to, if you were designing on a search algorithm and you are add to look at different options of whether you can save on the close list or whether you can save on the open list, what would be your choice. If somebody says that you can, I will give an algorithm in which, the open list sizes is minimize, made constant or something like that or made linear like, I D A star or it somebody says that, I will give you an algorithm, in which the close list side is ((Refer Time: 26:14)) reduced, which one would you choose. Would you rather proven the open list or would you rather proven the close list?

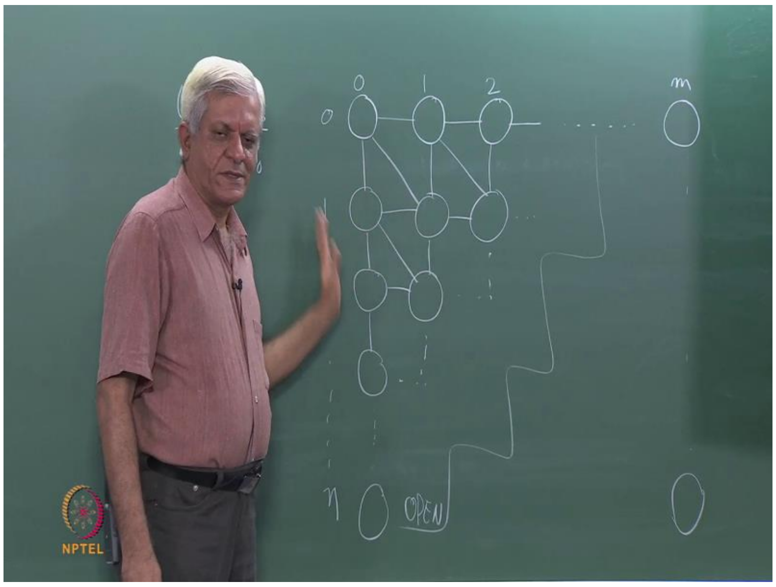
Student: ((Refer Time: 26:26)) size of depends on the size of the grid the number of nodes.

Well, I take your answer to mean it depends on the topology of the problem essentially.

But, if you look at a general case, were every node has b successors, you know branching factor is b, every node as b successors. So, very often the community tends to even if you have a graphs from which you are searching, then you know, if you remember the branch and bound, we started off with, in which the duplicates were not remove and the same node would appear and different parts of the tree. Many people tends to think of that is space, as a space of searching over paths, because each node in a different part of the tree depict the different path, because you know from route it as a different path. Then of course, it tends to go very highly.

So, when we studied D F I D, we had argued that the number of nodes in the last layer of breadth first search by itself was must larger than all the internal nodes seen. And then we argued that, because if that is only the extra work it we are doing and you are getting linear space in D F I D are oppose to exponential space of breadth first search. Then you are saving on the open list largely, that is what D F I D was doing saving on the open list essentially. But, there are problem sometimes, when you want to save on the close list, when the close list can become a greater problem than an open list.

(Refer Slide Time: 28:48)

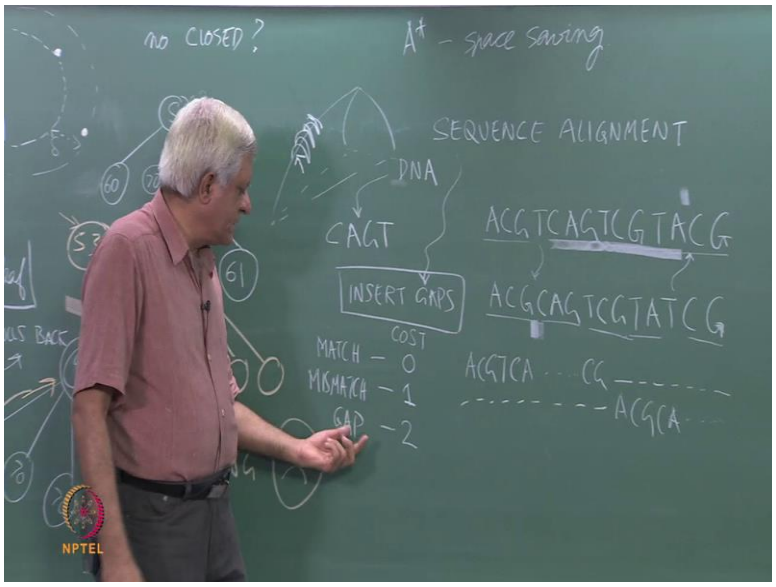


And this problem that we are looking at, this city route finding problem is such a problem essentially. Now, if you visualize, how search will progress in this space. This will be the start node, it has three successes, it would generate those three successors; it will take one of them, generate its three successors and so on and so forth, that is how search will progress. How will the search frontier look like or the boundary of the search look like? It would be some of which look like this. At some level, the search frontier would look like; it would have gone some steps to this thing and some steps down at different parts of the space and search frontier look like.

So, for such a problem how in terms of m and n, how is this open list, search frontier is equal to the open list. So, this is the open list, how the open going in terms of m and n, the father you go away from the… So, the distance from the source is m plus n or i plus j if you want to say, if you have a gone i steps down and j steps to the right, it is a i plus j. The open list is only growing linearly, as you go further and further away. In the worst case, when you are just about to pick the last node, open list would be m nodes there, and n nodes here essentially, m plus n essentially. Whereas, the close list, which is all the nodes that we have traverse inside this area is growing quadratically essentially.

Because, it is kind of is like an area is suppose through line that you are drawing like that. So here, is one problem with the close list is growing faster than the open list, the difficulty in solving this problem comes when the combinations which you have, there if you want to go from this node to some node here, there are you can go like this; or you can go like this; or you can go like this; or you can go like this. So, there are many different combinations and that is what gives rise to the exploding search spaces. So, there is some ((Refer Time: 30:35)) trying to save on the close list essentially.

(Refer Slide Time: 30:54)



So, let me introduce to a problem which has in the last 10 or years or maybe 15, 20 years has become very important, and that is a problem of sequence alignment. And it is a problem, which has spade work in search, search had become a kind of dormant area in a research, but because of this new problems, which are coming out. People were set of motivated into devising better algorithms essentially. So, if you look at for example, D N A sequence alignment, and that is a problem with in bio informatics, for example, you want to do very often essentially.

So you do, if you want to do for example, genome sequencing and thinks like that, the way the sequence genome is that they get bits and pieces of the sequence from different parts. And then they have to assemble the whole genome sequence by aligning, you know, if there are two sequences which have a partial overlap. Then if they can align that, then they can reconstruct ((Refer Time: 31:54)). So, will take a simpler version of this problem, so the alphabet of this is let us see C A G T, which are this four chemicals, which make a pure D N A, and you have sequences of these characters.

So, let us say, there is one sequence which is like this, A C G T C some arbitrary sequence I have written here. And let us say there is another sequence, if you look at it carefully, they are not identical, I made some small changes in the two sequences, if they were identical then the problem of alignment is straight forward, you just put once, you can against the other sequence. If they are not identical, the two sequences and that includes the possibility that they are of different lens also, I may you this sequence, I may give you some another sequence like, T A C G then if I say, can you align this sequence with this sequence, you might say that yes, I can take this part here, and alignment this part here and have an alignment essentially.

But, what about sequences like this, you can see that, I can match A with A; C with C; G with G; but now I have T here and a C here essentially, what do I do essentially. So, in sequence alignment, you are allowed to insert gaps, if inserting the gap improves, the rest of the sequence alignment, then you allowed to insert the gap. So, you can see that, that once I align A C G with A C G here, then if I insert the gap here, which means, I will align this T with a gap. So, let us say this transfers that then I can again align C A G here, with C A G here, T C G, T C G everything is getting aligned, up to this T A.

And then suddenly, I have to insert another gap, so I insert one gap here, and then I can ((Refer Time: 34:34)). So, if you can make out this diagram, I am inserting a gap here, in this sequence and I am inserting a gap here in this sequence and by doing so, I am improving the alignment essentially. So, the problem of sequence alignment is to find, some alignment, which is optimal according to some criteria will define that in the moment, which is the best essentially. Observe that, I could have simplify said like this, that take this whole sequence, and the place it with gaps all the way, and then start this sequence.

So, I could have then something like this, A C G T C A up to C G here, and then started the next sequence from here, A C G C A and so on, and these are all gaps. That is of course, a simple algorithm, take the first sequence, align it with gaps, then take the second sequence and insert gaps in the first sequence in that place; obviously, that is not a good alignment essentially. So, how do we differentiate between good and bad, we give some cost to every operation, and let us say we do the following in practice, people follow probably more discriminative cost function, but will follow simple cost function, which says that matching.

So, if there is a match, then cost is 0, so if I am aligning in A with an A, I am paying a 0 cost, mismatch cost 1. So, if I am aligning a T with a C for example here, then I am paying a cost of 1 and a gap, I have to pay a cost of 2. So, let us say that, for example if there is only one character, which is different, if let us say only this T and this, instead of this C there was a... let us say another C here or something like that or a A or a G, which is not T, and the rest was the same. Then I could get away by paying a cost of mismatch, which is 1, rather than insert two gaps to you know, take care of those things essentially.

But, in a situation like this, where you can see that, inserting one gap here produces a match for so many more characters, simply by inserting one gap here. So, I am the cost I am paying by inserting this gap, I am regaining by getting 0 costs for all the rest essentially. So, obviously there is a notion of optimality here and we want to find a sequence alignment, which has optimal cost based on these three, this. In practice of course, you may have a cost matrices which would say that, aligning a C with A has a certain cost, aligning a C with a G has a certain cost and so on and so forth or mismatches may have you know different cost.

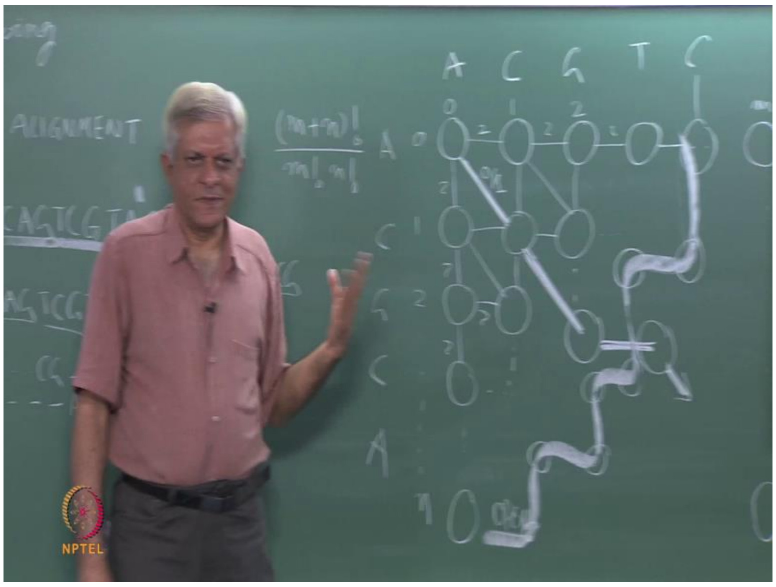
We are just assuming that all mismatches have 1 cost and gap filling has 2 costs. So, how does one solve this problem, what would be a good algorithm to do sequence alignment?

Student: dynamic program

Dynamic programming. So, they say they if to use the animal programming, till this bio informatics people came in and then suddenly, we have sequences of hundreds of thousands of characters and those algorithms, so good is essentially failed essentially. So, if you do some kind of an analogy, if you would see the dynamic programming is like branch and bound that we would, that we were studying essentially in some sense essentially. So, you are looking at this graph

Student: you can use the word as the rows and column and if cost will diagonal ((Refer Time: 38:59)) so if you move long diagonal ((Refer Time: 39:02)) and machining it.

(Refer Slide Time: 39:11)



So, let us try that, this is by first sequence here A C G T C. So, I will just use the few characters A C G T C. So, I will draw this here, and I will draw this here. And let me take five characters from here, A C G C A, A C G C A or maybe I should, I have started labeling from the second listing, any way let us assume that, we are working here. So, now you can see that a diagonal move in this. So, if this I look at this alignment, matching A with A, C with C, G with G and T with C.

So, let us forget the first ones, I should have really drawn this like that, but this move amounts to saying that I am matching C with this C, then I am matching this G with this G, but then I run with T with ... So, I want to insert a gap here, in the second this, which means I am not going to traverse a character, which means I am not going to traverse down ((Refer Time: 40:28)). So, the next move would be like this and then I would be in this node, and then I would continue like this.

So, can you see that, the first move aligns C with C this move, the second move aligns G, it going from here; I am going from C to G and C to G here, so it aligns that. The third move I am going from G to C, I am not going from G to C, I am staying in G, but I am going from G to T here, which is like inserting a gap in this essentially. Of course, I am not shown mismatches here, but you can see that the cause of diagonals would be either

0 or 1, depending on whether the two characters that you are moving on are same or different.

So, if I move with A to A cost is 0, if I move with C to C cost is 0, from move with G with G, the cost is 0, if I would align T with C and moved diagonally, then the cost would be if 1. But, instead I am saying, I am going to insert a blank here, which is the horizontal move here and the cost is 2. So, essentially you can actually forget about alignment at this point and remember that the cost of every horizontal, a vertical move is 2. And the cost of a diagonal move is either 0 or 1, depending on whether a moving on a same character.

So, remember that moving down means, moving in this string; moving horizontally means moving in the other string and moving diagonally means moving on both strings and if you are moving on the same character then cost is 0, if you are moving on a different character, cost is 1. So, we are transforming this problem of sequence alignment into a graph search problem, where you have to go from this corner of the graph to the other corner of the graph with a optimal solution cost essentially. So, any question about this.

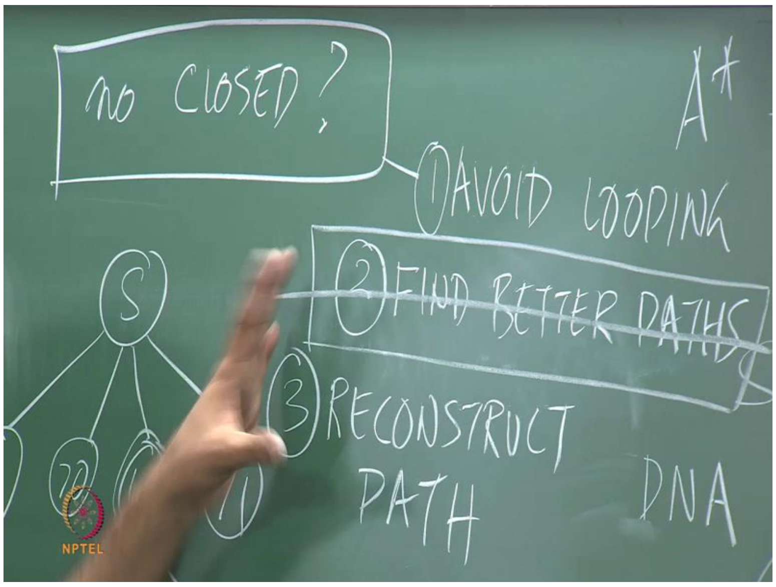
So, this has been, this has been a motivating problem for A I search community to look for newer and better problem, better ways of solving search problems. And why do we need, what do we mean by newer and better ways? We should be able to handle such graphs which have, let us say hundred thousand nodes in this direction and hundred thousand nodes in the vertical direction. And then you can imagine that and A star like algorithm will quickly run out of space essentially. So, the motivation here is to say one space essentially.

And here, I have just shown two strings, you can extend this to multiple strings assignments, alignments and you can imagine a third dimension coming here and then a four dimension and then a fifth dimension , you can align multiple strings. So, the graph would become multidimensional in nature and then you would have to find from one corner to the opposite corner essentially. So, the problem is have been transformed into a graph search problem and the goal is to save one space essentially.

So, we will begin with talking about saving on close list and again, the motivation is the same graph. Because, as you can imagine, if this was to be my search frontier which means, this is on open, this is on open, this is on open, this is on open or whatever, this is on open, this is on open, this is on open. If this was the nodes which are an open then the size of search can be utmost m plus n. So, it is going to grow linearly, at least it will be the smaller of m and n, as it goes down.

Initially of course, it will be very small, but the size of closed is going quadractically, because that is would like the area, which is enclosed in this korf, which is m into n in the worst case at that point, when you reach there. If you have see explode the entire graph and we want to see. So, this is the motivating problem where, we would be happy to say one closely essentially. So, let us just ask the question, what is close doing for us in search, before we move on to the algorithms which. So, this question, no closed, supposing, I am did not have closed, what would happen to my problem.

(Refer Slide Time: 45:16)



So, what is the purpose, what is the function that a functionality of close list in a search algorithm one is avoid looping. Now, this motivating example may not allow for looping, because we have said the edges are directed, but in general of course, looping is possible. So, we will take a more general view, two is find better path. So, for example, in ((Refer Time: 45:51)) star algorithm or in A star algorithm, you keep the nodes in the closed and you may find a better path them in which case, you update that path and that kind of a thing.

And the third thing, one more things closed as for us. In general, if you look at search in a planning problem, what do you do, if goal test is true?

Student: ((Refer Time: 46:30))

Closed has all the information about parent pointers, which allows us to reconstruct the path essentially. So, the third thing it does is, it allows us to reconstruct the path. So, if we are going to think of having an algorithm without closed, then we would have to worry about, these three issues or these three things that closes doing for us essentially. So, let me first begin with the second one, how can we get around this problem that, I do not need to maintain close with one path and then find a better path later and I do not want to do all this.

Student: ((Refer Time: 47: 23))

((Refer Time: 47:26)) star algorithm maintains a value, but in the closed or in the colored node as they call it, but it may find a better path, no it may not find a better path. In ((Refer Time: 47:36)) star algorithm, when once it colors a path or once it puts it in close either always found the best path essentially. But, in A star also we can do that and the we had discussed that, when in A star can you say, that if you have put a node in closed, you have already found the best path to the node.

And you would not find a better path later, when does that happen, come on we just did it. I mean not just now, but in the recent past, when the monotone condition, when the heuristic function satisfies the monotone condition, we proved that, whenever node is picked for inspection, which means it is put in to close. A star has already find, found the optimal cost of that node essentially. So, this updating of past we do not have to worry about that, what we do have to worry about is avoid looping and reconstruct path.

So, we will do that in the next class, we will look at some algorithms for which drastically proven the close list. After that, we will look at some algorithms, which proven the open list of course, we have seen I D A star is one such a example, which proves a open list, but we will look at a other variations. And then we will try to see whether, we can have an algorithm which proven both the list essentially. We shall take us to the state of art till about 2005, when that last paper was published I think. So, I will stop here and then we will take those algorithms of in the next class essentially.