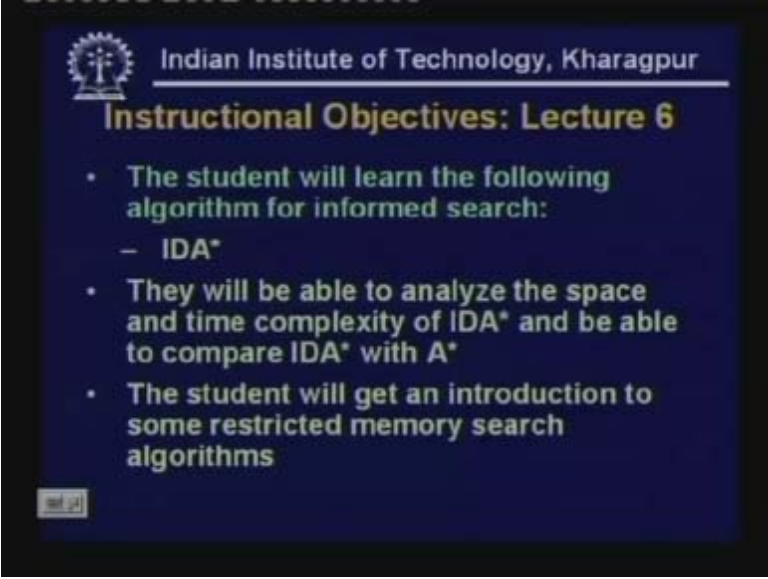


Artificial Intelligence
Prof. Sudeshna Sarkar
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecture - 6
Informed Search -2

Today we start our 6th lecture. This is the 2nd lecture on Informed Search. In the last class we looked at the different properties of A star. Today we will continue with A star and talk about another algorithm called IDA star.

(Refer Slide Time: 01:35)



Indian Institute of Technology, Kharagpur

Instructional Objectives: Lecture 6

- The student will learn the following algorithm for informed search:
 - IDA*
- They will be able to analyze the space and time complexity of IDA* and be able to compare IDA* with A*
- The student will get an introduction to some restricted memory search algorithms

The instructional objectives of today's lecture are the following:

In this lecture the student will learn the algorithm IDA star which is an algorithm for Informed Search. They will be able to analyze the space and time complexity of IDA star and they should be able to understand the difference between IDA star and the algorithm A star. The student will also get an introduction to some of the restricted memory search algorithms like RBFS that is recursive best first search as well as MA star. We will also discuss briefly, some local search algorithms like hill climbing as also simulated annealing. The students will get a brief idea about these algorithms. They will also have an understanding about the properties of search algorithms.

(Refer Slide Time: 02:35)

Indian Institute of Technology, Kharagpur

A* Search

- Hart, Nilsson & Rafael 1968
 - Best first search with $f(n) = g(n) + h(n)$
 - Where $g(n)$ = sum of edge costs from start to n
 - And $h(n)$ = estimate of lowest cost path $n \rightarrow$ goal
 - If $h(n)$ is admissible then search will find optimal solution.

Underestimates cost of any solution which can be reached from node

Let us just quickly go over A star search. So A star is a best first search where we use the function $f(n)$ is equal to $g(n)$ plus $h(n)$ to order the nodes for expansion. The nodes are expanded according to their increasing f values. We have also seen that in case $h(n)$ is admissible then search will find the optimal solution. A heuristic function $h(n)$ is said to be admissible if it underestimates the cost of any solution which can be reached from the node.

(Refer Slide Time: 03:30)

Indian Institute of Technology, Kharagpur

Algorithm A*

OPEN = nodes on frontier.
CLOSED = expanded nodes.
OPEN = {<s, nil>}
while OPEN is not empty
{

}
return failure

So, to briefly recapitulate algorithm A star we have two lists OPEN and CLOSED.

If we want to deal with a state space containing repeated states and we want to ensure that a particular state does not get expanded multiple times we need to keep along with the OPEN list all other lists as CLOSED which consists of nodes which have already been expanded. OPEN is the list of nodes on the frontier of the search tree.

Initially CLOSED is empty and OPEN has a single node the start node and the path associated with the node is an empty path. The algorithm A star executes this loop. While OPEN is not empty certain things are done. So in this loop that we will see in the next slide we try to see if we can find the path from the start node to a goal node. If we do not find such a path in this loop and we have no more lists no more nodes to expand. That is, OPEN becomes empty, the algorithm should return failure. Now this is what we do in the body of this loop.

(Refer Slide Time: 04:54)

```
remove from OPEN node  $\langle n, p \rangle$  with min  $f(n)$ 
place  $\langle n, p \rangle$  on CLOSED
if  $n$  is a goal node, return success (path  $p$ )
for each edge  $e$  connecting  $n$  &  $m$  with cost  $c$ 
    if  $\langle m, q \rangle$  is on CLOSED &
        $\{p|e\}$  is cheaper than  $q$ 
       then remove  $m$  from CLOSED,
          put  $\langle m, \{p|e\} \rangle$  on OPEN
    else if  $\langle m, q \rangle$  is on OPEN &
        $\{p|e\}$  is cheaper than  $q$ 
       then replace  $q$  with  $\{p|e\}$ 
    else if  $m$  is not on OPEN put  $\langle m, \{p|e\} \rangle$  on OPEN
```

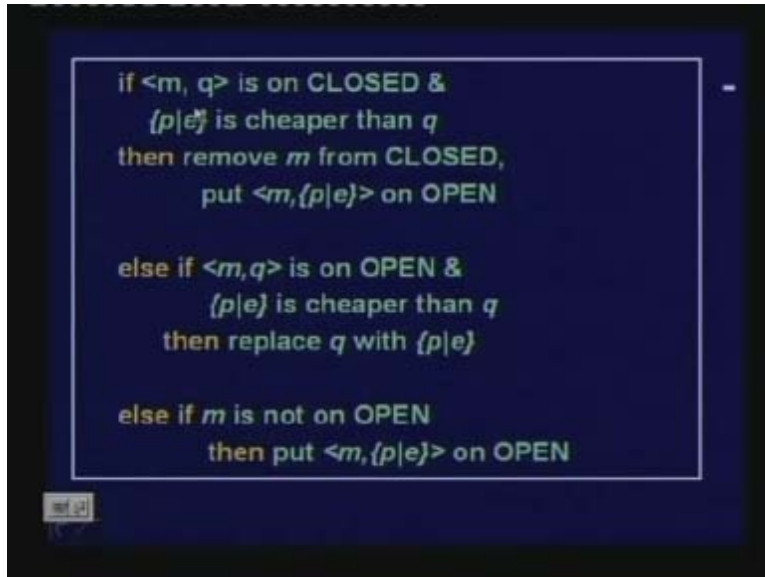
We remove from OPEN the node n, p whose $f(n)$ value is minimum. So n is the node, p is the path associated with the node, n is that node whose $f(n)$ value is minimum, we put n, p on CLOSED because it has been expanded. If n is a goal node we return success and we return the path p which is the path associated with this loop. Otherwise we find all edges coming out of the node n that is edges connecting node n and with some other node m .

Suppose there is edge e connecting n with m whose cost is c . Now we consider this state m . If m is already on CLOSED with the path q and our current path p concat e happens to be cheaper than the cost of the path q then we will like to consider this current path as the best path [here](#).

If q is cheaper we throw away our recorder path. But if m, q is on CLOSED and the current path is cheaper than q we remove the node m from CLOSED and put the new node the current node whose state is m whose path is p concat e we put this node at

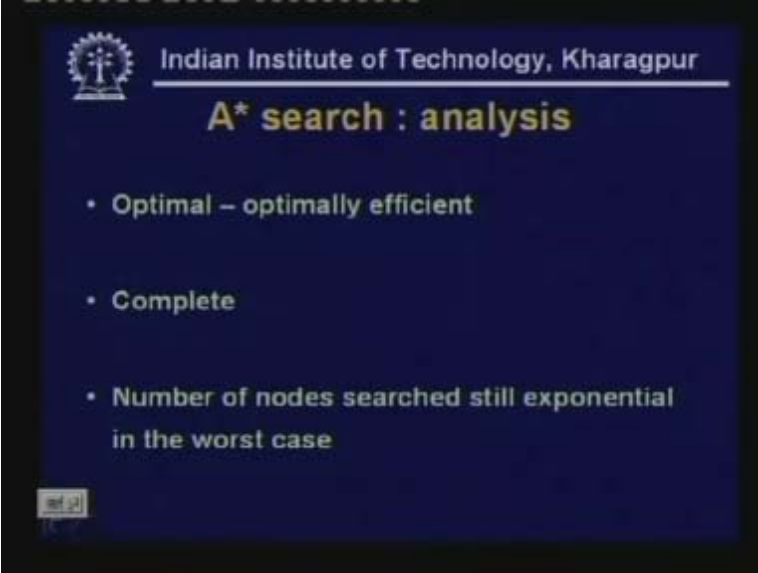
OPEN. Otherwise if it is the case that a node m is on OPEN already with a different path q and the current path cost is cheaper than the cost of q then we replace q with the current path p concat e . Otherwise if m is not on OPEN and not on CLOSED we put m, p, e on OPEN. So this algorithm is expanded in this section.

(Refer Slide Time: 07:05)



In the last class we have done an analysis of the algorithm A star and we have seen that A star is an optimum algorithm. That is, if there is a path from the start state to the goal state then A star is guaranteed to find the optimum path and optimum path to the goal node. We have also seen that all algorithms that use the same heuristic function $h(n)$ A star is an optimally efficient algorithm. A star is also complete. That is, it finds a path if one exists. And the number of nodes searched by A star is exponential in the worst case.

(Refer Slide Time: 08:06)

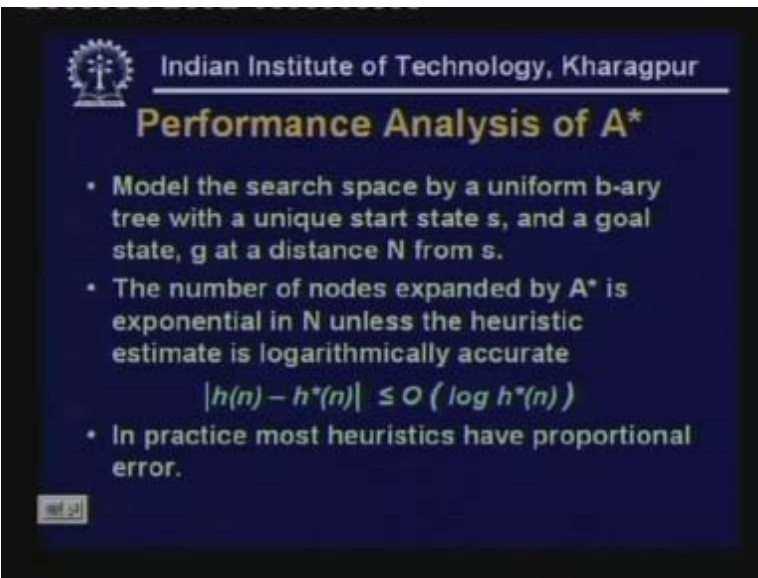


Indian Institute of Technology, Kharagpur

A* search : analysis

- Optimal – optimally efficient
- Complete
- Number of nodes searched still exponential in the worst case

(Refer Slide Time: 08:10)



Indian Institute of Technology, Kharagpur

Performance Analysis of A*

- Model the search space by a uniform b-ary tree with a unique start state s, and a goal state, g at a distance N from s.
- The number of nodes expanded by A* is exponential in N unless the heuristic estimate is logarithmically accurate

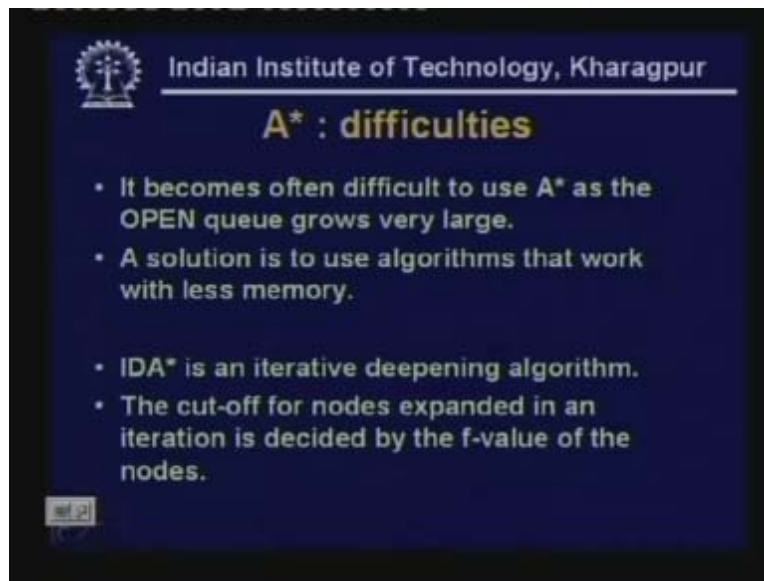
$$|h(n) - h^*(n)| \leq O(\log h^*(n))$$

- In practice most heuristics have proportional error.

So, if we model the search space and assume that the search space is a uniform b r e tree and there is a unique start state s and one goal state g whose distance from the start state s is N and every arc has a cost of 1 then we can show that the number of nodes expanded A star is exponential in n unless the heuristic estimate is logarithmically accurate. That is, suppose A star n is the actual cost from n to the goal, the optimum cost of n to the goal and suppose h(n) is our estimate. If h(n) minus h star n is less than or of the order of log of h star n then the heuristic is said to be logarithmically accurate. And in this case A star has a polynomial complexity on such a search tree.

However, in most cases we can say utmost that practically the heuristic functions we can get are utmost proportional to A^* . The error is proportional. Error is not logarithmic and in those cases A^* has exponential complexity. So A^* has exponential complexity. As we have seen A^* tries to improve upon general breadth first search by using this heuristic function to restrict the number of nodes being expanded.

(Refer Slide Time: 10:21)



However, this restriction is not good enough because the complexity of A^* is still exponential. If we get a better heuristic function then the number of nodes in expansion can become less. So A^* still can take a lot of time on a reasonably large problem. But even worse than the time complexity of A^* is the fact that it becomes difficult to use A^* even before the time runs out because the size of the OPEN queue becomes too large because the space complexity of A^* is the space required for storing the OPEN queue. And in those cases where we use the CLOSED queue we also have to store the CLOSED.

Now the solution so that A^* does not have this bottleneck or such algorithm does not have this bottleneck is to have a search algorithm which uses less space. Now we will discuss an algorithm IDA star which is Iterative Deepening algorithm. And IDA star requires less space. We have already talked about Iterative-Deepening search which is basically search which does a level by level expansion of the search tree. And we have seen that Iterative-Deepening depth first search has a space complexity of order $b \cdot d$ where b is the branching factor and d is the depth of the tree.

Here d is utmost the N which is the cost of the solution. So, if we use Iterative Deepening search we can have an algorithm which works in linear space. However, Iterative Deepening A^* is actually slightly different from Iterative-Deepening search. It is actually very similar where the nodes to be expanded in a particular iteration depend of the f value of the nodes and not on the level of the nodes. So, in Iterative-Deepening A^* the cutoff for nodes expanded in iteration is decided by the f value of the nodes.

Therefore we fix a limit of the f value and in an iteration we only expand those nodes in a depth first manner whose f value is less than or equal to this f limit. In the next iteration we increase the value of this limit. **Now let us look at Iterative-Deepening A star.**

(Refer Slide Time: 13:22)

Indian Institute of Technology, Kharagpur

Iterative-Deepening A*

- Like iterative-deepening depth-first, but...
- Depth bound modified to be an f -limit

1. Start with $\text{limit} = (\text{start})$
2. Prune any node if $f(\text{node}) > f\text{-limit}$
3. Next $f\text{-limit} = \text{min-cost of any node pruned}$

The graph shows six nodes: a, b, c, d, e, and f. Node 'a' is the start node. Edges connect (a,b), (a,e), (b,c), (c,d), (c,e), and (e,f).

In Iterative-Deepening A star is a depth first Iterative-Deepening search but the depth bound used is a f limit. So we start with the initial limit as the f value of start. We use a f value of start as the initial cutoff or initial limit. Then with that limit we do a depth first search. And in doing this depth first search we prune any node if f of that node is greater than the value of f limit. So, after the current iteration the next f limit is chosen to be the minimum cost of any node that was proved.

Now let us try to illustrate Iterative-Deepening A star on the following graph. We have six nodes a b c d e f. Initially we fix a f value equal to the f value of the start node. Suppose the start node is a and suppose the f value of the start node is 15 so using the f limit as 15 we do a depth first search in which a, b and e are expanded. Now c and f were the nodes which were pruned in this first iteration and suppose out of f value of f and f value of c the f value of c was minimum and it was 21 And at the next iteration we will use the f value of 21 and we will do a depth first search with this f limit equal to 21. And in the next iteration c was the node pruned with a particular f value.

We will set that equal to the f limit and do the next iteration and so on. So this is the schemata of Iterative-Deepening A star. Basically it is Iterative Deepening search where search is pruned when the f value of a node exceeds the current f limit. And the minimum f value of a node pruned is used as the next value of f limit. Now let us look at the properties of the algorithm IDA star.

(Refer Slide Time: 16:13)

Indian Institute of Technology, Kharagpur

IDA* Analysis

- Complete & Optimal
- Space usage \propto depth of solution
- Each iteration is DFS - no priority queue!
- # nodes expanded relative to A*
 - Depends on # unique values of heuristic function

IDA star is complete and it is optimal. It is complete because it will find a solution if one exists. It is optimal because the first time it expands a goal node it would have found an optimum solution. The space usage is proportional to the depth of the solution. Each iteration of IDA star is a depth first search so it requires only space equal to b times d , the branching factor times the depth of the tree so space requirement is linear.

Now you note what the extra nodes are which are expanded in every iteration. They are those nodes whose f value is immediately just greater than the f value of the nodes in the previous iteration. So the number of nodes expanded in subsequent iterations is the nodes with increasing f values. Hence IDA star is an optimal algorithm. It is obvious that IDA star expands more nodes than A star because certain nodes are expanded more than once.

How many more nodes IDA star expand compared to A star?

That depends on the number of unique values of $f(n)$. So, if it is the case that between two iterations the extra number of nodes expanded is large then IDA star is fairly efficient in the number of nodes expanded. But consider the case when every node has a unique f value. In that case at every iteration only one new node will be expanded and in this case IDA star will expand the square of the number of nodes in A star. So space required by IDA star is $O(bd)$, number of nodes expanded relative to A star.

(Refer Slide Time: 18:23)

Indian Institute of Technology, Kharagpur

IDA* Analysis

- Space required : $O(bd)$
- # nodes expanded relative to A^*
 - In 8 puzzle: few values \Rightarrow close to # A^* expands

The number of iterations = the number of distinct f-values.

In 8 puzzle there are few values of f and the number of nodes expanded by IDA star is very close to the number of nodes A^* expands. Suppose in 8 puzzle we use a Manhattan distance or the number of misplaced tiles as the heuristic function the number of such values such a heuristic can take is very small, the number of misplaced tiles can be utmost 8. So the total f value is actually also not very large. As a result the number of unique m values is small. So IDA star tends to expand a large number of extra nodes in every iteration. And the number of iterations is equal to the number of distinct f values.

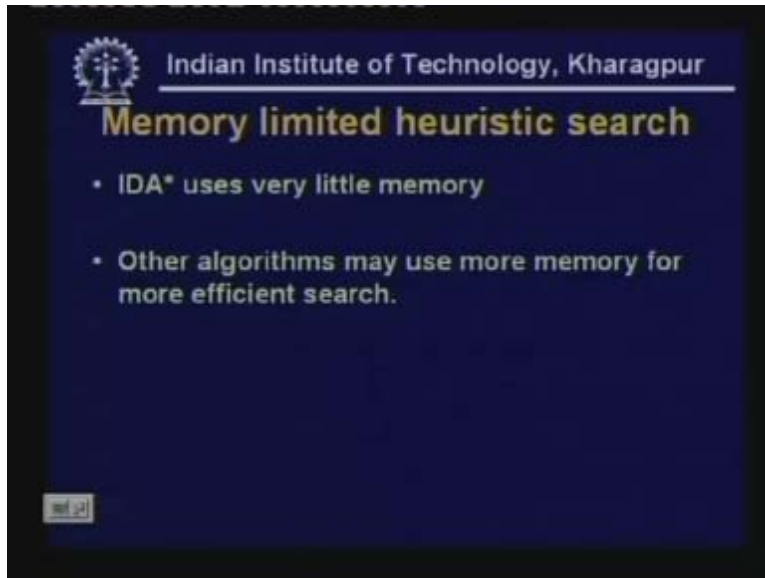
However, consider the traveling salesperson problem where we have a graph of n cities and the objective of the graph in salesperson is to start from a city traverse all the cities exactly once and come back to the starting city. Imagine that every distance between a pair of cities is a different real value. In this case it is possible that each f value is unique. So IDA star will expand one new node in every iteration and the number of nodes expanded would be the first iteration 1 second iteration 2 third iteration 3 and in the last iteration the number of nodes expanded will be equal to n . So the total number of nodes expanded equal to $O(n^2)$ where n is the number of nodes A^* expands. So we see that IDA star can expand utmost a quadratic number of nodes expanded corresponding to A^* star.

However, in other situations like 8 puzzle IDA star would expand only a constant factor more nodes than A^* star would expand. Also, we see that because in IDA star basically we are using a depth first search we cannot avoid repeated node expansion so IDA star does expand duplicate nodes in cyclic graphs.

If a graph contains cycles IDA star will not be able to detect the cycles and it will expand those nodes repeatedly. But the main advantage of ID star is its linear memory. For example, 15 puzzle is a problem similar to eight puzzle except that the problem is on a 4 into 4 grid, there is one empty square. So if you run A^* star on 15 puzzle it will take a

huge amount of time to solve and a huge amount of memory. And 24 puzzle possibly you cannot solve in A star using normal resources because of the huge space requirement and time. However, you can use IDA star which does not need to keep this OPEN queue which will be able to solve 8 puzzle and 15 puzzle. IDA star uses very little memory, it only uses linear memory and it may expand up to n^2 nodes if A star expands n nodes.

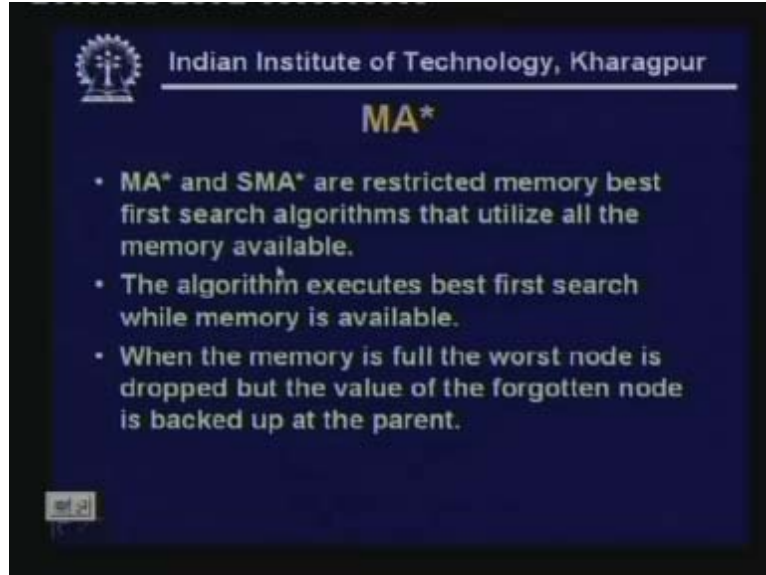
(Refer Slide Time: 22:18)



The idea of memory limited heuristic search algorithms is to use the available memory optimally so as to have a more efficient algorithm which reduces node expansion. Several such algorithms have been proposed and we will briefly mention these algorithms.

Recursive Breadth First Search or RBFS is an algorithm actually uses only linear space but it is often better than IDA star because it mimics best first search. RBFS keeps track of the f value of the best alternative path available from any ancestor of the current node. So whenever RBFS is a linear memory algorithm it does not keep track of all the nodes but those nodes that it has expanded and the algorithm chooses to forget. The algorithm backs up at a node the f value of the best successor of that node. So, at a particular node we have current f value and we also know what is the second best f value based on what we had expanded earlier. When the current f value this alternate f value then RBFS will explore this alternate path. Let us now see another algorithm MA star and its related algorithm SMA star.

(Refer Slide Time: 24:39)

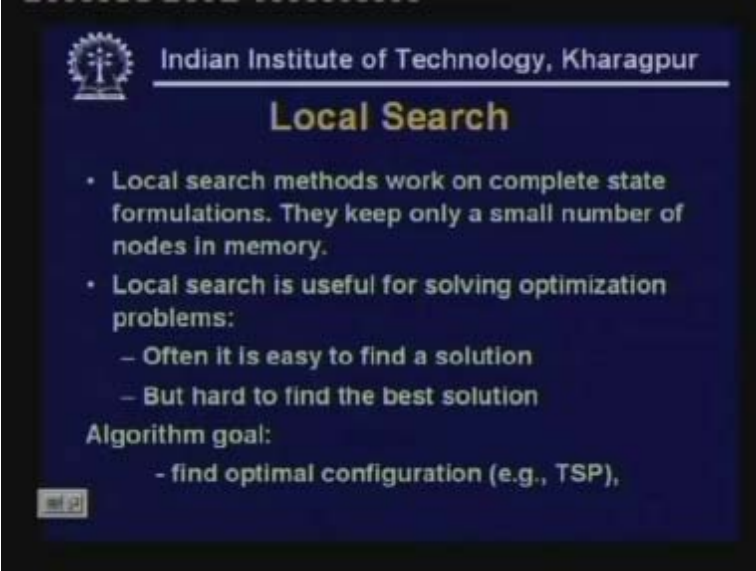


These are restricted memory best first search algorithms that utilize all the available memory. The algorithms execute best first search when memory is available. It does normal best first search. When the memory becomes full the worst node is dropped and the value of the forgotten node is backed up to the parent. When the node is dropped the value of this backed up node is backed up to the parent.

We have seen systematic search algorithms: depth first search, breadth first search, A star, IDA star, bidirectional search and so on. And we have seen that A star and IDA star and such algorithms or even RBFS, MS star use some heuristic function to restrict the number of nodes that are expanded. And the most natural types of heuristic functions we can obtain is that these search algorithms have exponential time complexity. There are situations where it is not possible to use such algorithms to get the best solution to a problem.

Local search methods are used in a class of search algorithms where we need not have an optimum solution. There are problems where we need a solution and we can get a solution which may not be the optimum solution but we can get the solution in limited time. Local search methods work on formulations where in every configuration we have a complete state. Therefore local search does a search on complete state formulations.

(Refer Slide Time: 26:17)



Indian Institute of Technology, Kharagpur

Local Search

- Local search methods work on complete state formulations. They keep only a small number of nodes in memory.
- Local search is useful for solving optimization problems:
 - Often it is easy to find a solution
 - But hard to find the best solution

Algorithm goal:

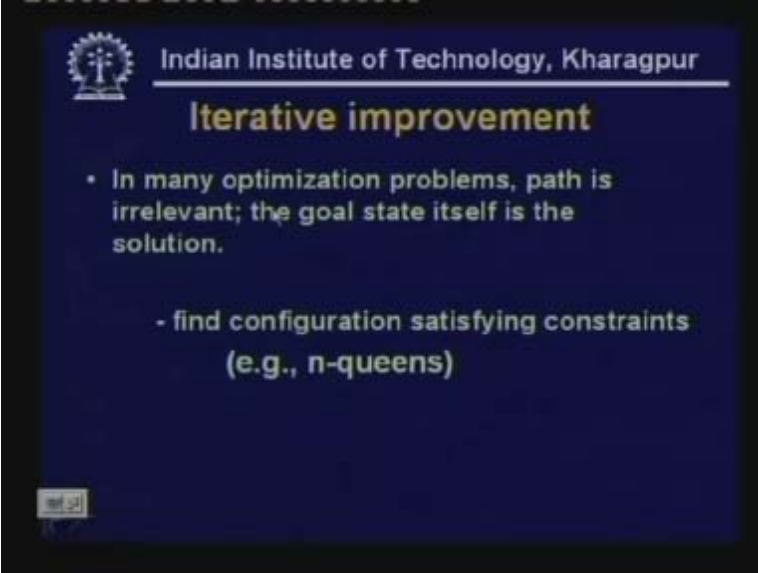
- find optimal configuration (e.g., TSP),

In local search typically constant number of nodes is kept in memory and these nodes are all configurations of the entire state. These states are **perturbed** to get the next state. Local search is useful for two types of problems. One it is useful for solving optimization problems in those optimization problems where it is often easy to find a solution but hard to find the best solution.

Consider the traveling salesperson problem in a fully connected graph. It is easy to get one solution to the problem because any permutation of the city is a solution but it is difficult to get the best solution because there are n factorial such permutations and we do not know one efficient way of finding the best permutation in polynomial time. So, in TSP the objective is to find the optimum configuration which is difficult to find. We do not know to find a good algorithm but if we just want to find a solution then it is easy.

Many local search algorithms use iterative improvement and we can try to get a solution and get improvements to that solution. Secondly, for these optimization problems like TSP we do not have to keep track of the path to the solution. When we have the solution we immediately know the path. In TSP we do not have to keep track of the path. Local search is ideal for such cases. In problems like 8 puzzle the final state does not give us any information about the path. The solution path has to be obtained. Local search is not very good for such problems. But for problems like TSP where the path is not important local search is very useful.

(Refer Slide Time: 28:26)



Indian Institute of Technology, Kharagpur

Iterative improvement

- In many optimization problems, path is irrelevant; the goal state itself is the solution.
 - find configuration satisfying constraints (e.g., n-queens)

Another example of a problem is the n queens problem where we have a n by n chess board and we have to put n queens on the board so that none of the queens are attacking each other. For this problem once we have the final configuration we know the solution. So n queens problem is not obvious how we can get a solution. So we can define n queens as an optimization problem. So the basic idea of the iterative methods is to start with the solution and improve it so that we can get a better solution.

(Refer Slide Time: 30:14)



Indian Institute of Technology, Kharagpur

Local Search

- Iterative methods
 - Start with a solution
 - Improve it towards a good solution

Example is the n queens problem. We have to put n queens on the chess board. So in this case we have to put 4 queens. What we can do is that we can start with any placement of

the 4 queens on the board. But if you place the queens like this obviously there are many conflicts.

(Refer Slide Time: 30:32)

Indian Institute of Technology, Kharagpur

Iterative improvement example:

- N queens
- Goal: Put n chess-game queens on an $n \times n$ board, with no two queens on the same row, column, or diagonal.

The slide displays three 4x4 chessboards connected by arrows, illustrating a sequence of moves to reduce conflicts. The first board shows a configuration with 4 conflicts (two queens on the same diagonal and two on the same column). The second board shows a configuration with 2 conflicts (two queens on the same diagonal). The third board shows a configuration with 0 conflicts (no two queens on the same row, column, or diagonal).

We can reformulate our objectives as saying we want to have a board with 0 conflicts. Or we can say we want to minimize the number of conflicts. That is we want to minimize the number of pairs of queens which can attack each other. So we start with this state where this pair is attacking each other, this pair is attacking, this pair attacking, this pair is attacking, this pair is attacking. And then we move to this configuration where this is attacking, this is attacking, this is attacking. And then we move to this configuration where this pair is attacking each other and none of the other pairs are attacking each other. So we have reduced the number of conflicts between queens. So, hill climbing is one local search method. It is also called gradient ascend or gradient descent. In hill climbing the idea is that, we start with a current configuration and we move to a new configuration to maximize the value.

For example, in n queens our objective is to minimize the number of conflicts. So we start with an arbitrary configuration and we move to a new configuration where the number of conflicts is reduced. So we find from the current configuration different configurations of different neighbors and we move to that neighbor which has the minimum number of conflicts. So we move to the neighbor which is best. That is why it is called hill climbing. We go to the best neighbor and we continue until we get to a state where all the neighbors are worse than this state.

We can talk about hill climbing when we are either minimizing a value function like minimizing the number of conflicts or maximizing the value function, they are symmetric. For example, in 8 queens we can start with an initial state and we can define our successor function as moving a queen to another square in the same column. And the

cost associated with the configuration is the number of pairs of queens that are attacking each other.

(Refer Slide Time: 33:08)

Indian Institute of Technology, Kharagpur

Hill climbing - example

- Complete state formulation for 8 queens
 - Successor function: move a single queen to another square in the same column
 - Cost: number of pairs that are attacking each other.
- Minimization problem

We want to minimize these pairs. This is a schematic diagram of the state space of hill climbing. We are starting at a particular position and we want to move to a neighbor whose value is large.

(Refer Slide Time: 33:30)

Indian Institute of Technology, Kharagpur

Hill climbing

- Problem: depending on initial state, may get stuck in local extremum.

Global maximum

Local maximum

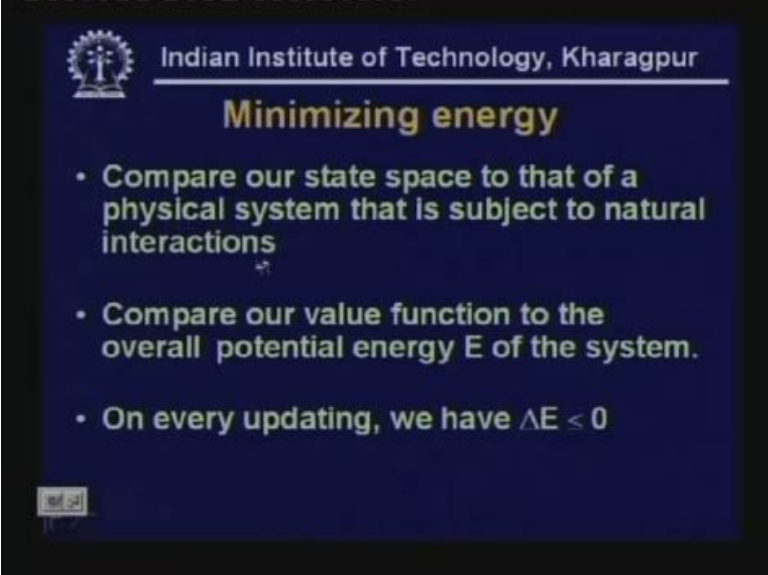
value

states

Now depending on which initial state we start from we can get stuck in local extremum. Suppose we start from this position and then we find that this is the best neighbor and

then we find this is the best neighbor so we can move until we get here whose neighbors are all worse than this. This is the solution that we have obtained and this also happens to be an optimum solution. But suppose you start from here and then if you move to the best neighbor and then its best neighbor and then here you will get to this position where all its neighbors are worse but nevertheless it is not the optimum solution to this problem. So, if we use hill climbing because it is a local search method we can get stuck at a local extremum instead of all this landing up at the global optimum.

(Refer Slide Time: 34:41)



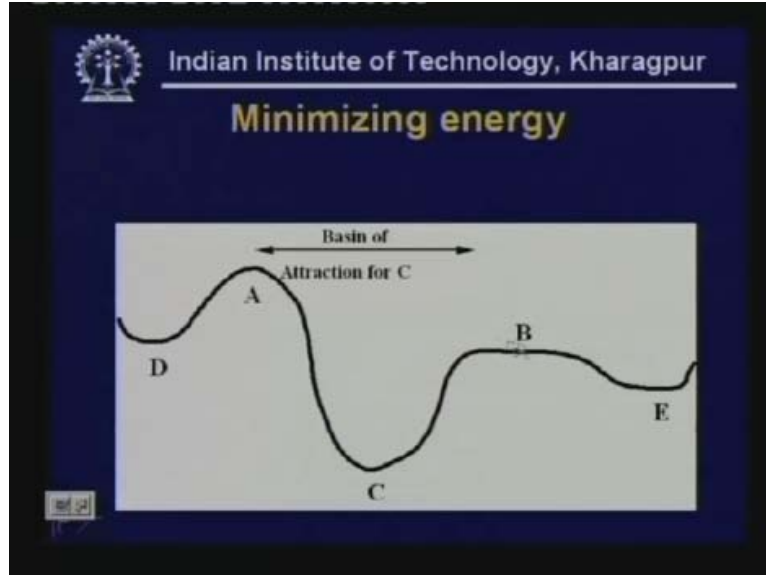
Indian Institute of Technology, Kharagpur

Minimizing energy

- Compare our state space to that of a physical system that is subject to natural interactions
- Compare our value function to the overall potential energy E of the system.
- On every updating, we have $\Delta E < 0$

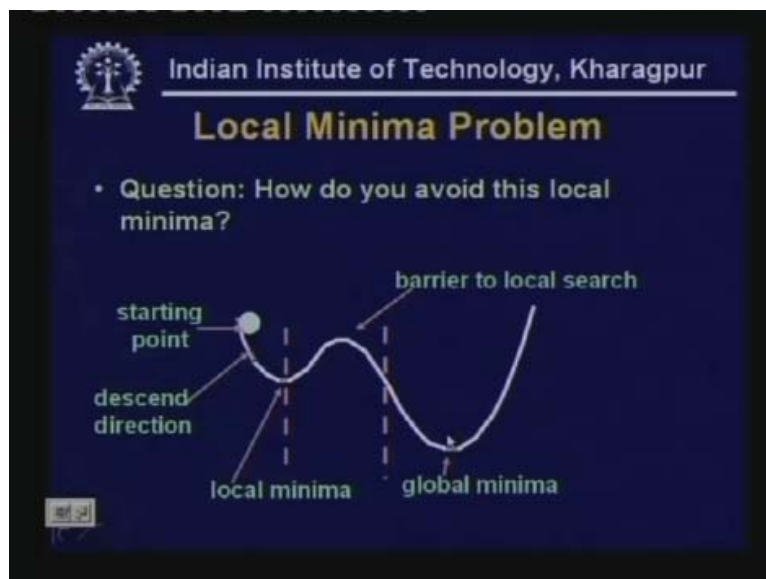
Hence, we can compare our state space to that of a physical system that is subject to natural interaction and we can look at the analogy of our value function to the overall potential energy of the system. On every updating we have $\Delta E \leq 0$. So this is the change in potential energy.

(Refer Slide Time: 35:08)



Now in this schema of the value of the state space in this state this is our global optimum. This diagram is for a minimization problem where c is the global minimum and what we see is that, if we start at any of these positions we can get to the global minimum. So this region is the region of attraction the basin of attraction for c. If our initial starting is between these two lines then we can reach this global minimum. However, if our initial starting position is somewhere here we cannot reach this global minimum but we will instead settle to a local minimum. Therefore in local search we cannot guarantee global minimization because of existence of many local minima in general.

(Refer Slide Time: 36:57)



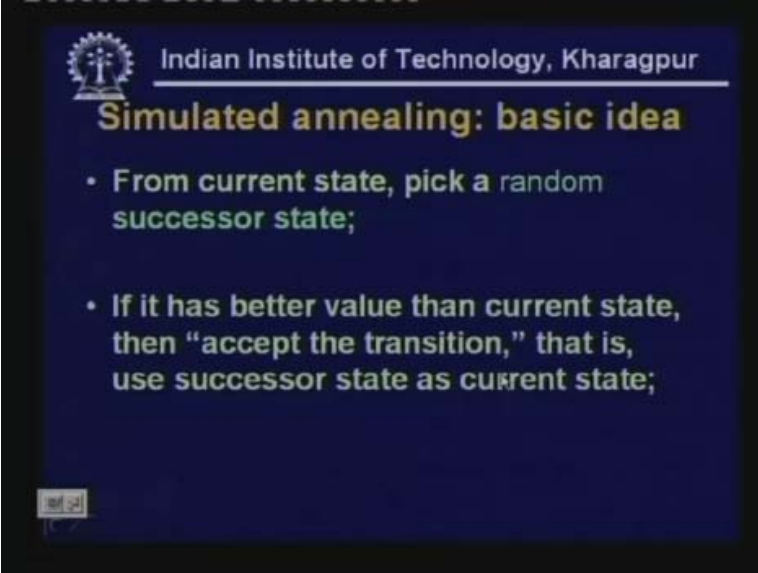
Now the question is, so if you use hill climbing you cannot avoid this problem how you can at all avoid this problem. Let us look at an example: Suppose this is our value function at different states and this is the starting point. Now if you start at this starting point and we use hill climbing or hill descending this is the direction of steepest descent and this is where our ball will roll until it settles at the local minimum. So this hump is acting as a barrier to the local search and our ball is settling at the local minima instead of going into the global minima.

(Refer Slide Time: 37:30)



Now, if you want to avoid this problem we must let our ball also go in the sub optimum direction in order to get to the local minimum. To the global minimum it may be necessary to climb the hill at certain points even though you want to reach the deepest valley. So, occasional ascents are required so that the ball can get over this hump and settle at a minimum which may be global or local. Therefore, ascent will help escape the local optima. But if we allow ascent may also help the ball get past the global optima after reaching it. We can avoid this by keeping track of the best state. If you start from here and you also allow ascents and then we keep track of the best state reached so far we may be able to identify the best state which this ball has reached.

(Refer Slide Time: 38:45)



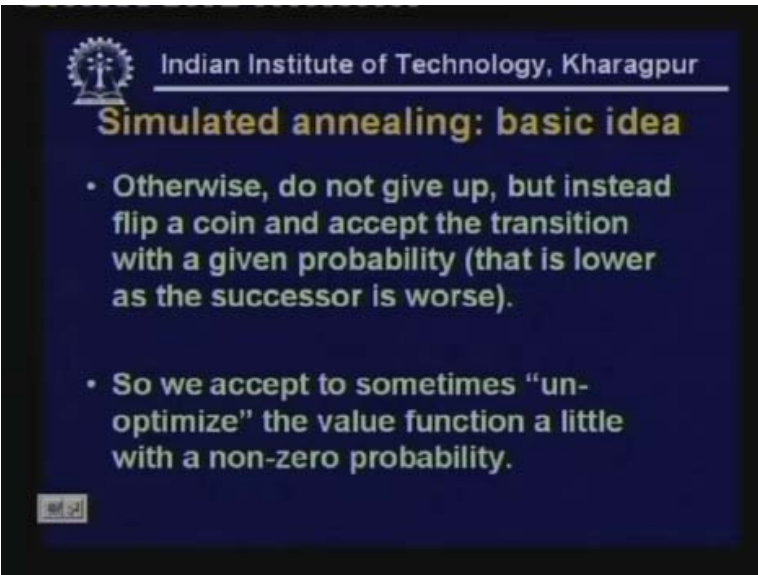
Indian Institute of Technology, Kharagpur

Simulated annealing: basic idea

- From current state, pick a random successor state;
- If it has better value than current state, then "accept the transition," that is, use successor state as current state;

Simulated annealing is an algorithm which avoids some pitfalls of hill climbing and the basic idea is this. From the current state we pick our random successor state. If that successor state has a better value than the current state then we accept the transition. If it has a worse value we do not give up but with some probability we accept a worse transition.

(Refer Slide Time: 39:21)



Indian Institute of Technology, Kharagpur

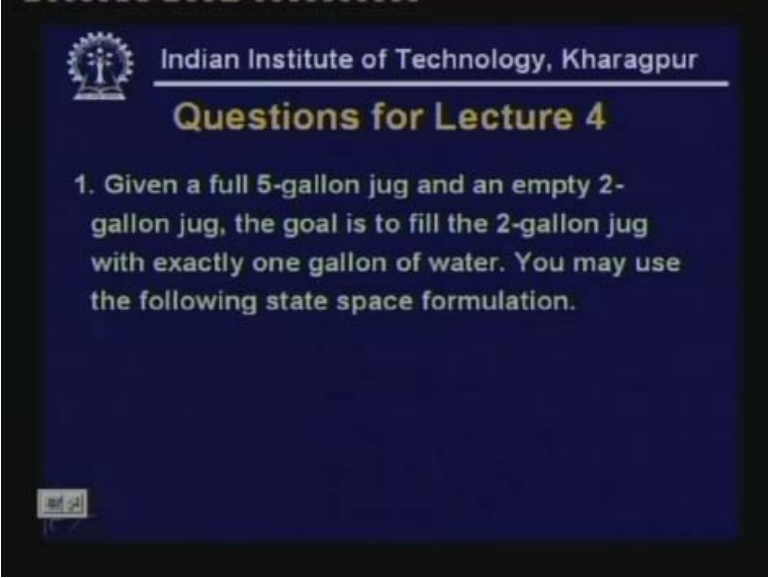
Simulated annealing: basic idea

- Otherwise, do not give up, but instead flip a coin and accept the transition with a given probability (that is lower as the successor is worse).
- So we accept to sometimes "un-optimize" the value function a little with a non-zero probability.

If we get a successor whose value is worse we flip a coin and accept the transition with some probability. Sometimes we accept non optimal solutions. Initially with a higher probability we accept non optimal solutions. As time passes with smaller probability we

accept non optimum solutions so that we allow our ball to settle at a minimum. There are other local search algorithms like genetic algorithms.

(Refer Slide Time: 40:28)



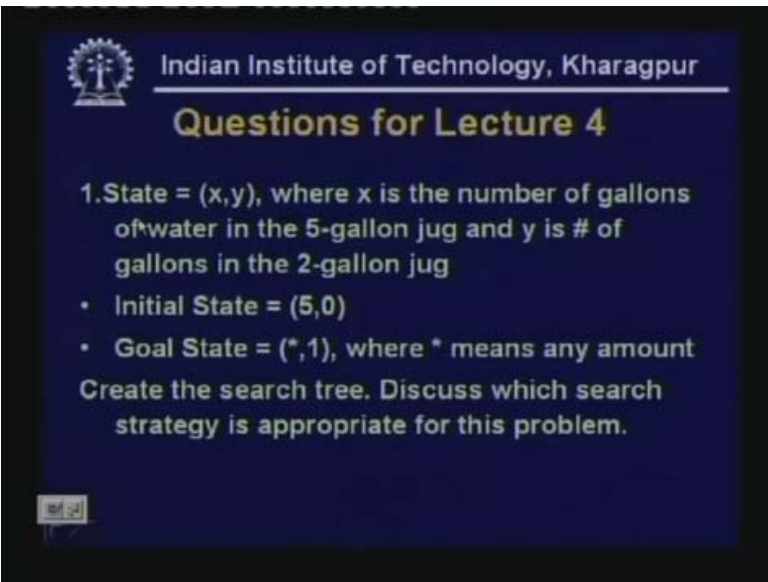
Indian Institute of Technology, Kharagpur

Questions for Lecture 4

1. Given a full 5-gallon jug and an empty 2-gallon jug, the goal is to fill the 2-gallon jug with exactly one gallon of water. You may use the following state space formulation.

What we will do now is discuss the solutions to the questions of lecture 4. This was the question. You are given a full 5 gallon jug and an empty 2 gallon jug. Your objective is to fill the 2 gallon jug with exactly 1 gallon of water.

(Refer Slide Time: 40:33)



Indian Institute of Technology, Kharagpur

Questions for Lecture 4

1. State = (x,y) , where x is the number of gallons of water in the 5-gallon jug and y is # of gallons in the 2-gallon jug

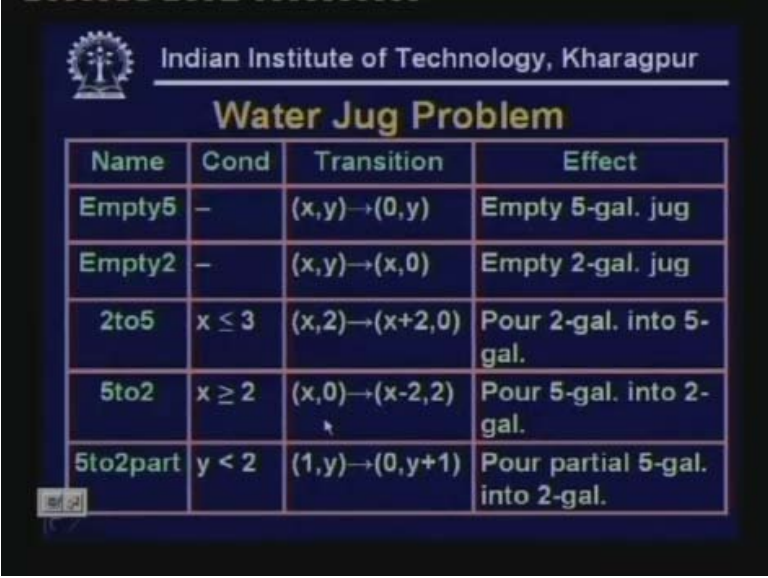
- Initial State = $(5,0)$
- Goal State = $(*,1)$, where $*$ means any amount

Create the search tree. Discuss which search strategy is appropriate for this problem.

You use the state space formulation like this. A state is represented by a pair x, y where x is the number of gallons of water in the 5 gallon jug, y is number of gallons of water in

the 2 gallon jug. Initial state is first the jug contains 5 gallons and secondly the jug is empty so 5, 0. Goal state is (star, 1) that is the second jug must have 1 gallon. The first jug can have any value we do not care. For this problem you have to create the search tree and discuss which search strategy is appropriate for this problem. Let us look at the solution to this problem.

(Refer Slide Time: 41:15)



Indian Institute of Technology, Kharagpur

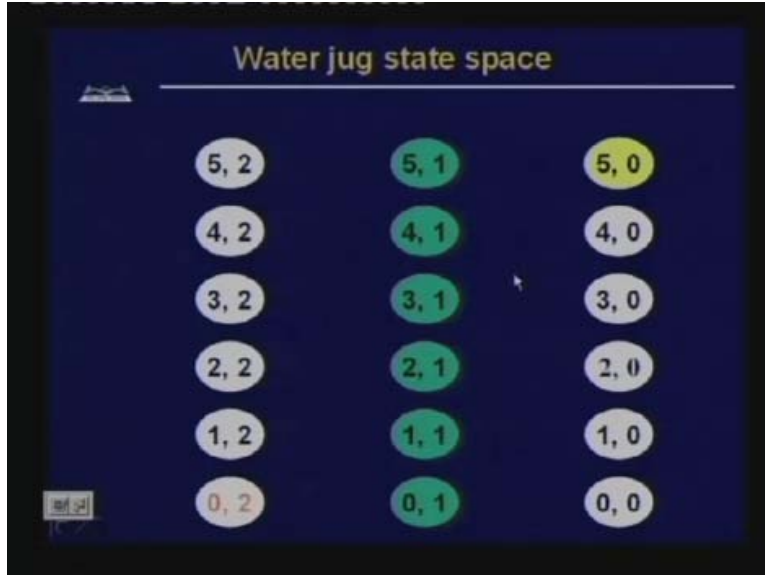
Water Jug Problem

Name	Cond	Transition	Effect
Empty5	-	$(x,y) \rightarrow (0,y)$	Empty 5-gal. jug
Empty2	-	$(x,y) \rightarrow (x,0)$	Empty 2-gal. jug
2to5	$x \leq 3$	$(x,2) \rightarrow (x+2,0)$	Pour 2-gal. into 5-gal.
5to2	$x \geq 2$	$(x,0) \rightarrow (x-2,2)$	Pour 5-gal. into 2-gal.
5to2part	$y < 2$	$(1,y) \rightarrow (0,y+1)$	Pour partial 5-gal. into 2-gal.

Now this table illustrates the different operators and the effect they have. These are the operators. Empty 5 gallon jug, empty 2 gallon jug, transfer from 2 to 5, transfer from 5 to 2, transfer from 5 to 2 partially, now empty 5 so these are the pre conditions. Empty 5 can be always used and as a result you move from the states (x, y) to the state $0 y$ because this jug becomes empty. Empty 2 that is the empty 2 gallon jug if you apply it on state (x, y) you get this state $(x, 0)$. And 2 to 5 is the operator to pour 2 gallon into 5 gallon.

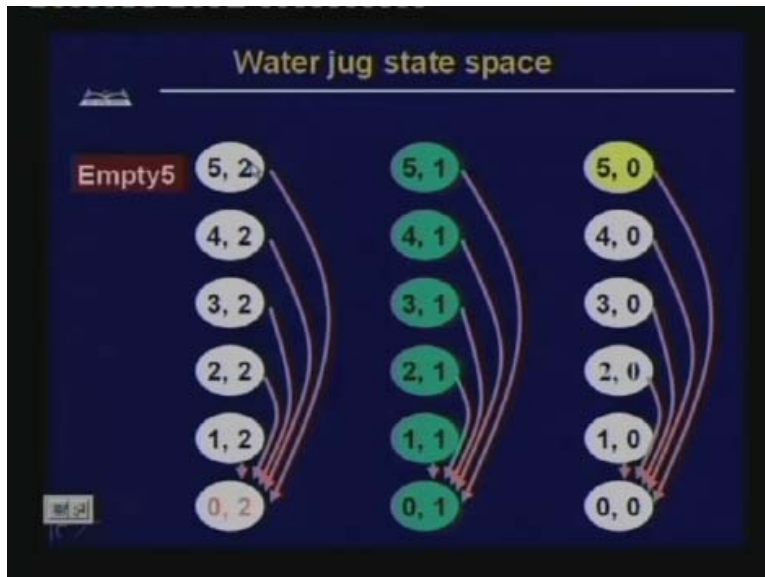
If you start with $(x, 2)$ you get to $(x \text{ plus } 2, 0)$ so pour 2 gallon into 5 gallon so this 2 gallon comes here. If you pour from 5 to 2 you can apply this only if x is greater than 2. That is, the 5 gallon jug initially has more than 2 gallons of water, in that case $(x, 0)$ will give you $(x \text{ minus } 2, 2)$. And 5 to 2 partially you can apply when y has less than 2 gallons of water. In that case $1y$ will become $(0, y \text{ plus } 1)$ so you pour the partial 5 gallon into 2 gallon.

(Refer Slide Time: 43:11)



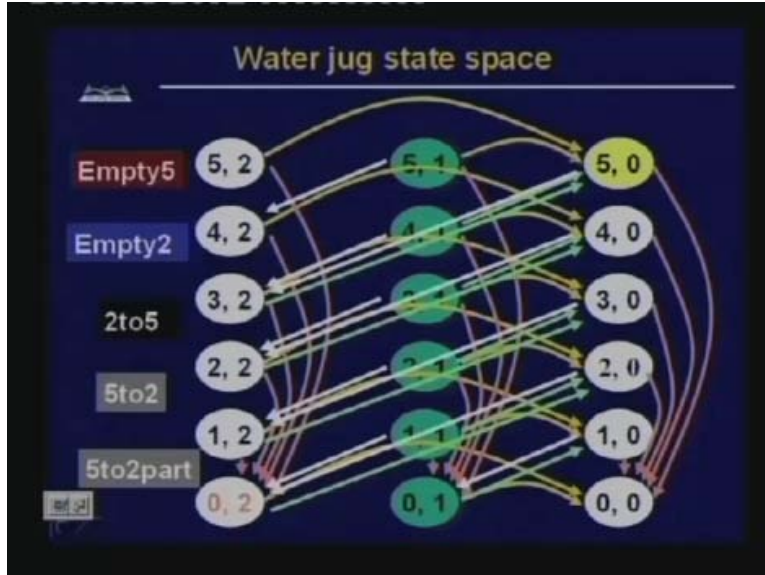
This table shows the effect of all the transitions. Now these are the different pairs of states 0, 0 0, 1 0, 2 1,0 1,1 1, 2 2, 0 2, 1 2, 2 all these are the states of our jug problem.

(Refer Slide Time: 43:25)



Now the empty 5 takes you from 5, 2 to 0, 2 from 4, 2 to 0, 2 2, 2 to 0, 2 and so on. It also takes you from 5, 1 to 0, 1 3, 1 to 0, 1 2, 1 to 0, 1 from 0 to 0, 0 and so on. The next operator is empty 2 which takes you from 5, 2 to 5, 0 4, 2 to 4, 0 5, 1 to 5, 0 3, 1 to 3, 0 and so on. And 2 to 5 takes you from 3, 2 to 5, 0 4, 1 to 5, 0 2, 2 to 4, 0 3, 1 to 4, 0 and so on. And 5 to 2 takes you from 5, 1 to 4, 2 5, 0 to 3, 2 4, 1 to 3, 2 and so on. So 5 to 2 part takes you from 2, 2 to 2, 0 2, 1 to 2, 0 and so on.

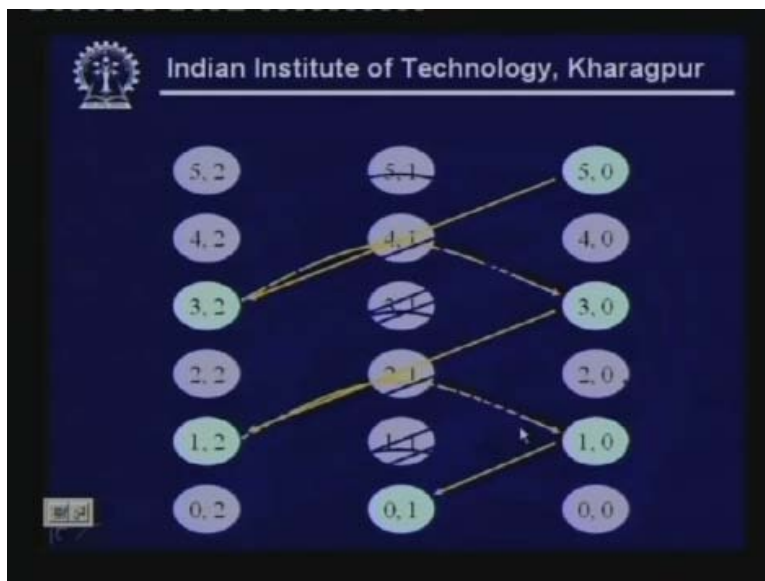
(Refer Slide Time: 44:36)



Now let us see how from this state space we have to find the solution from the starting state to a goal state?

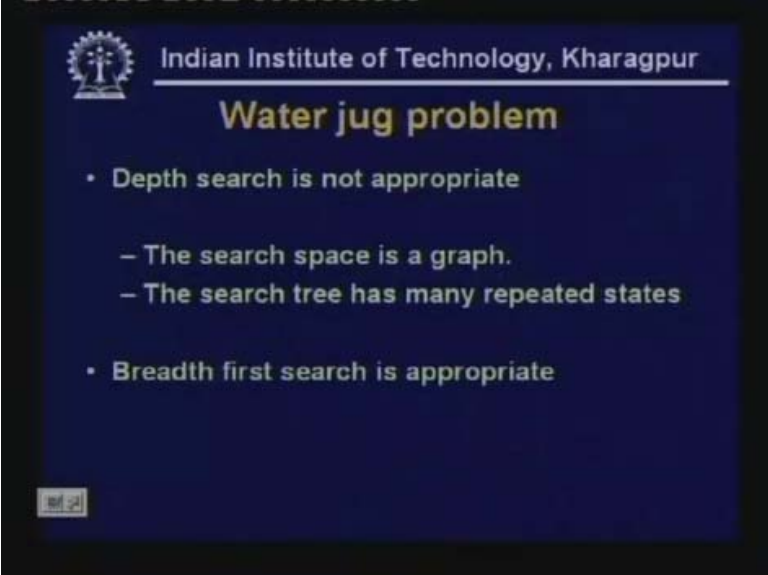
This is the state space we start with and this yellow box is our starting state. And now if we do a search we find that the optimum solution will go from 5, 0 to 3, 2 then from 3, 2 to 3, 0 from 3, 0 to 1, 2 and from 1, 2 to 1, 0 from 1, 0 to 0, 1. So, we have a solution which requires five steps.

(Refer Slide Time: 45:20)



Now let us see which search algorithm is appropriate for this problem. Depth first search is not appropriate because as you can see the state space is a graph and there are many repeated states so depth first search is not appropriate. Breadth first search is appropriate.

(Refer Slide Time: 45:43)



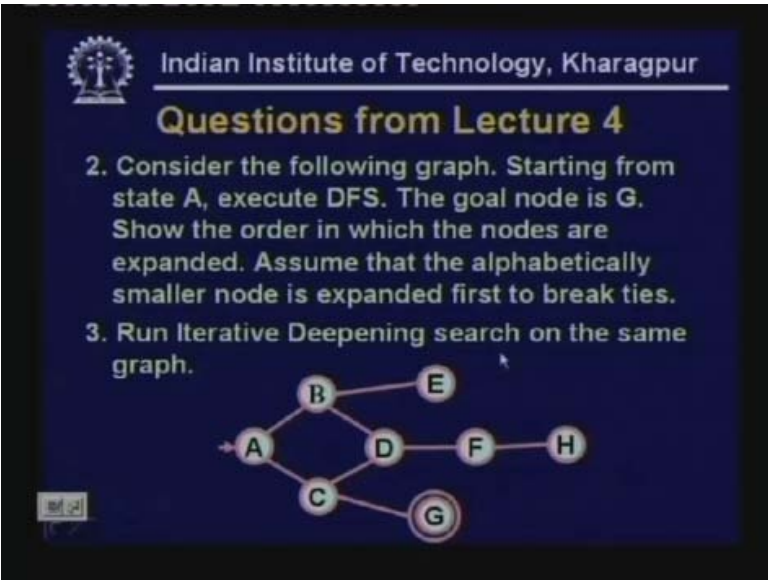
Indian Institute of Technology, Kharagpur

Water jug problem

- Depth search is not appropriate
 - The search space is a graph.
 - The search tree has many repeated states
- Breadth first search is appropriate

Now let us look at the second question from lecture 4. You are given the following graph: Starting from state A you have to execute depth first search and reach the goal node G and you have to show the order in which the nodes are expanded. And the third question was you run Iterative-Deepening search on the same graph.

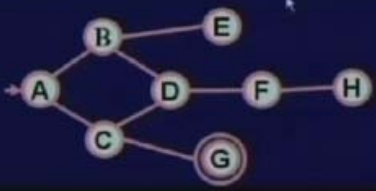
(Refer Slide Time: 45:49)



Indian Institute of Technology, Kharagpur

Questions from Lecture 4

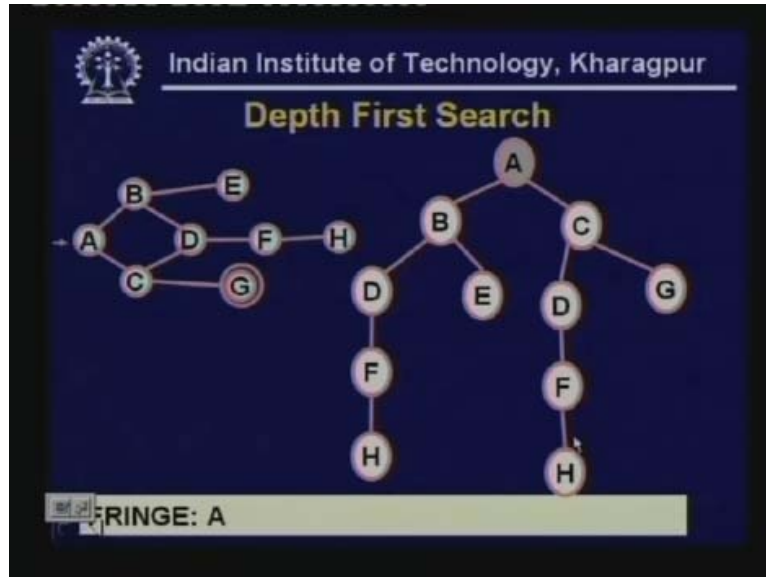
2. Consider the following graph. Starting from state A, execute DFS. The goal node is G. Show the order in which the nodes are expanded. Assume that the alphabetically smaller node is expanded first to break ties.
3. Run Iterative Deepening search on the same graph.



```
graph LR; A((A)) --- B((B)); A --- C((C)); B --- D((D)); B --- E((E)); C --- D; D --- F((F)); F --- H((H)); G((G))
```


We will see the solution to this problem. This is our graph and this is the search tree we obtained when we looked at these states.

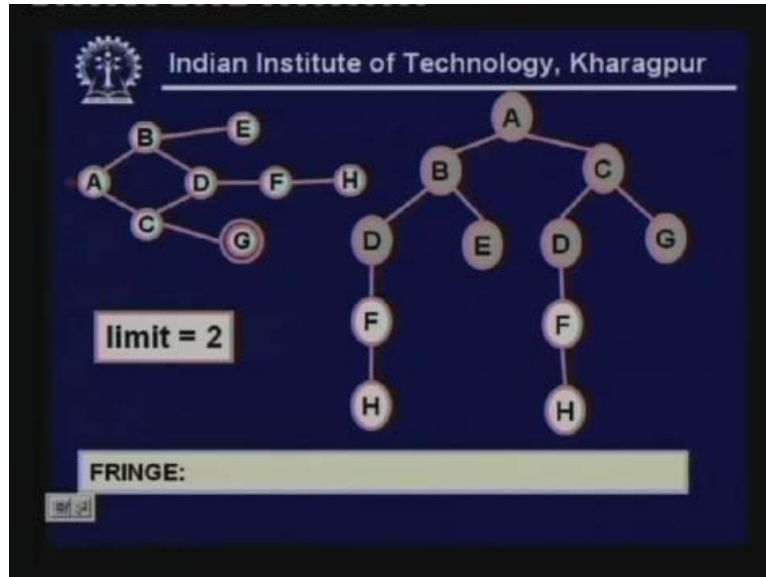
(Refer Slide Time: 46:21)



After we unfold this graph this is the tree we get. So, if you depth first search, initially our list OPEN, that is fringe contains the node A. Then A is removed from the fringe, B and C are expanded and put in OPEN. OPEN is a stack when we do breadth first search. Then B is removed from the front of the stack and D and E are added to the front of fringe. When the first node from fringe D is removed and a successor F is added to the front. Then F is removed from fringe its successor H is added, then H is removed and then E is removed. Then C is removed its successors D and G are added in the beginning then D is removed from the front its successor F is added, F is removed H is added H is removed G is removed and we have found a goal state. By depth first search this is the order in which the nodes are expanded A B D F H E C D F H and G.

Now let us run Iterative-Deepening search on the same state space. Initially the limit is 0 so we can expand A only. Then we set limit equal to 1 and we get this tree which is expanded in a depth first manner that is A then B then C. This illustrates the second iteration of IDS. In the third iteration limit is set to 2 and in this iteration this tree in this iteration this tree will be searched in depth first manner. And when we do search this tree we get, first A is removed from fringe and then B and C are added. B is removed D and E then D is expanded E is expanded C is expanded, D and G are added to fringe D is expanded and finally G is expanded and then we have found a goal node in this iteration where limit is equal to 2.

(Refer Slide Time: 48:49)



(Refer Slide Time: 49:15)

Indian Institute of Technology, Kharagpur

Questions for Lecture 5

1. Suppose you have the following search space:
 - a) Draw the state space of this problem.
 - b) Assume that the initial state is A and the goal state is G. Show how each of the following search strategies would create a search tree to find a path from the initial state to the goal state:

For lecture 5 these are the questions:
You will be given a following search space. This search space is specified by this state.

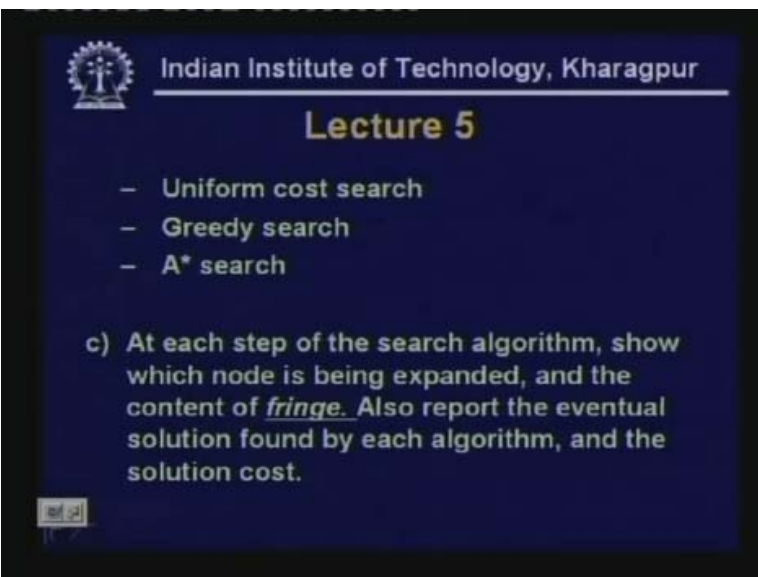
(Refer Slide Time: 49:25)



State	next	cost
A	B	4
A	C	1
B	D	3
B	E	8
C	C	0
C	D	2
C	F	6
D	C	2
D	E	4
E	G	2
F	G	8

In this state space there are seven states A B C D E F G. For state A there is a edge to state B with cost 4, there is an edge from A to C with cost 1, there is an edge from B to D with cost 3, B to E with cost 8, C to C cost 0, C to D cost 2, C to F cost 6, D to C cost 2, D to E cost 4, E to G cost 2 and F to G cost 8. So, given this state space you are required to draw the state space of this problem. We assume that the initial state is A and the goal state is G. For this state space you have to trace each of the following search strategies to create a search tree and find a path from initial state to the goal state.

(Refer Slide Time: 50:44)



Indian Institute of Technology, Kharagpur

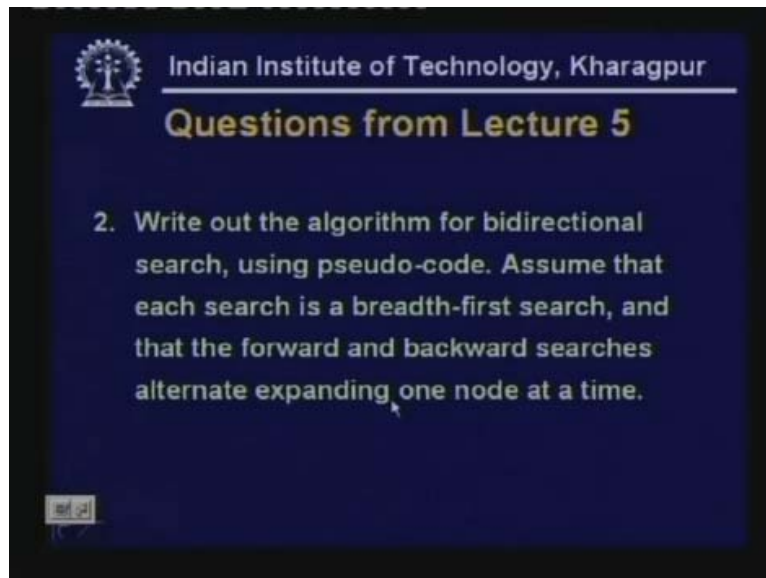
Lecture 5

- Uniform cost search
- Greedy search
- A* search

c) At each step of the search algorithm, show which node is being expanded, and the content of *fringe*. Also report the eventual solution found by each algorithm, and the solution cost.

So you are required to use this state space and run uniform cost search greedy search as well as A star search. For each of these three algorithms we will have their execution and find the order in which the nodes are expanded. At each step of the search algorithm you have to show which node is being expanded and the content of the OPEN list of fringe. You also have to report the eventual solution found by each of the algorithms and the solution cost that is obtained.

(Refer Slide Time: 51:24)



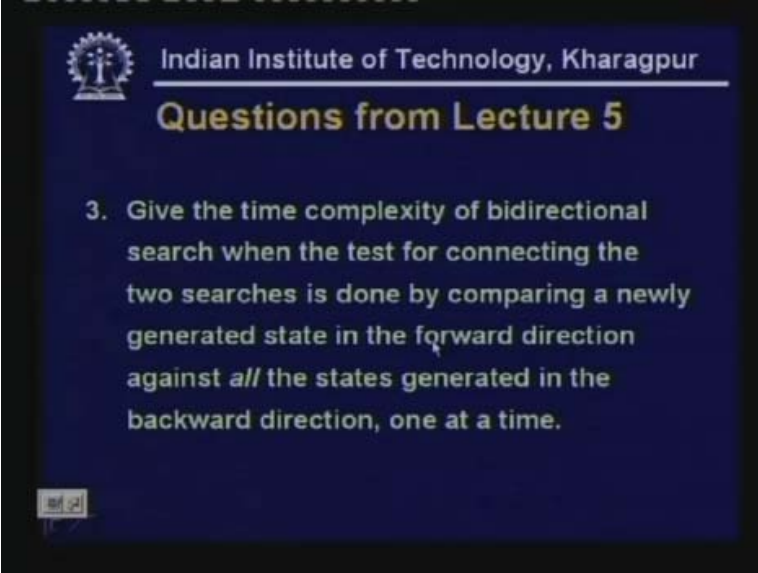
Questions from lecture 5:

You are required to write the algorithm for bidirectional search using pseudo code. We discussed bidirectional search in the last lecture but you will have to provide the pseudo code for this algorithm. You assume that each search is a breadth first search and the forward and backward searches alternate and they expand one node at a time.

Third question:

You have to find the time complexity of bidirectional search assuming that the test for connecting the two searches is done by comparing a newly generated state in the forward direction against all the states generated in the backward direction one at a time. So you remember in bidirectional search you have to check whether the two frontiers of the forward tree and backward tree are met. Suppose you do this check by checking the node expanded in the forward tree with all the nodes in the backward tree what would be time complexity of bidirectional search? That is question number 3.

(Refer Slide Time: 51:59)

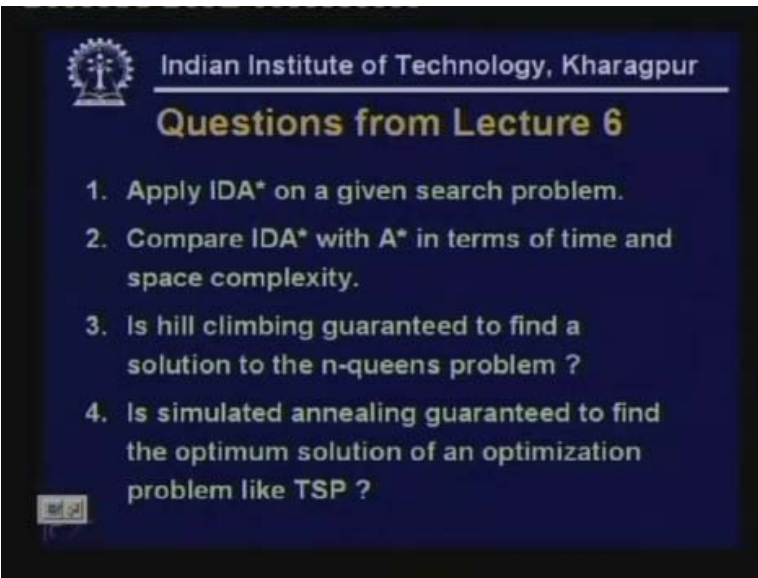


Indian Institute of Technology, Kharagpur

Questions from Lecture 5

3. Give the time complexity of bidirectional search when the test for connecting the two searches is done by comparing a newly generated state in the forward direction against *all* the states generated in the backward direction, one at a time.

(Refer Slide Time: 52:54)



Indian Institute of Technology, Kharagpur

Questions from Lecture 6

1. Apply IDA* on a given search problem.
2. Compare IDA* with A* in terms of time and space complexity.
3. Is hill climbing guaranteed to find a solution to the n-queens problem ?
4. Is simulated annealing guaranteed to find the optimum solution of an optimization problem like TSP ?

Now we come to the questions from the current lecture that is lecture 6. The problems for this lecture are of this type. You will be given a search space. You can use actually the same state space that we gave in question number 1 of lecture 5. In question number 1 of lecture 5 we specified the search space by this table. For the same search space you have to apply IDA star and for IDA star you will also require the value of the heuristic function. Assume some values of $h(n)$ and apply IDA star on this state space.

You have to compare IDA star with A star in terms of time and space complexity in general. So you compare the correct heuristics of A star with IDA star in terms of time complexity and in terms of space complexity, that is the second question.

Third question, suppose you are using hill climbing to find a solution to the n queens problem the question is, is hill climbing guaranteed to give you a solution?

Question number 4:

If you use simulated annealing are you guaranteed to find the optimum solution of the traveling salesperson problem? These are the questions of lecture 6.