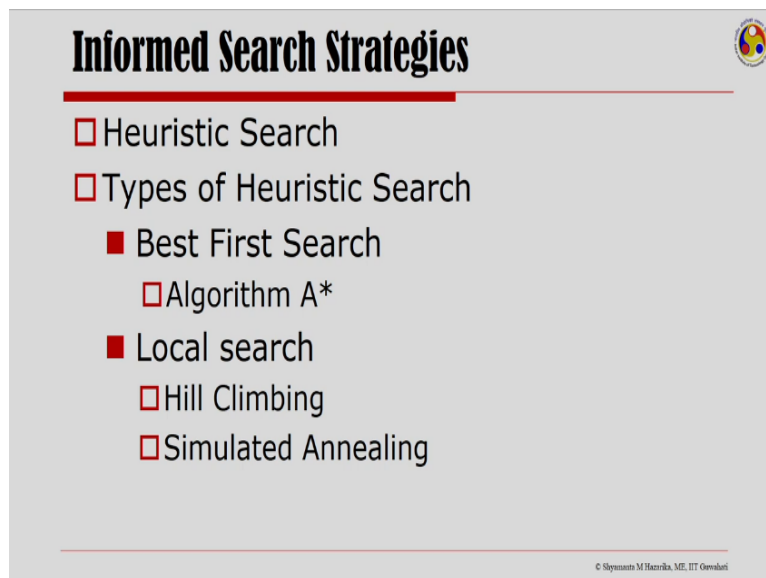


Fundamentals of Artificial Intelligence
Prof. Shyamanta M Hazarika
Department of Mechanical Engineering
Indian Institute of Technology - Guwahati

Module - 2
Lecture - 5
Informed Search

Welcome back to the course on fundamentals of artificial intelligence. We continue our discussion on search strategies. Having introduced uninformed search and heuristic functions, in this lecture we would cover informed search strategies. Informed search strategies use task-dependent information and are also called heuristic search.

(Refer Slide Time: 01:06)



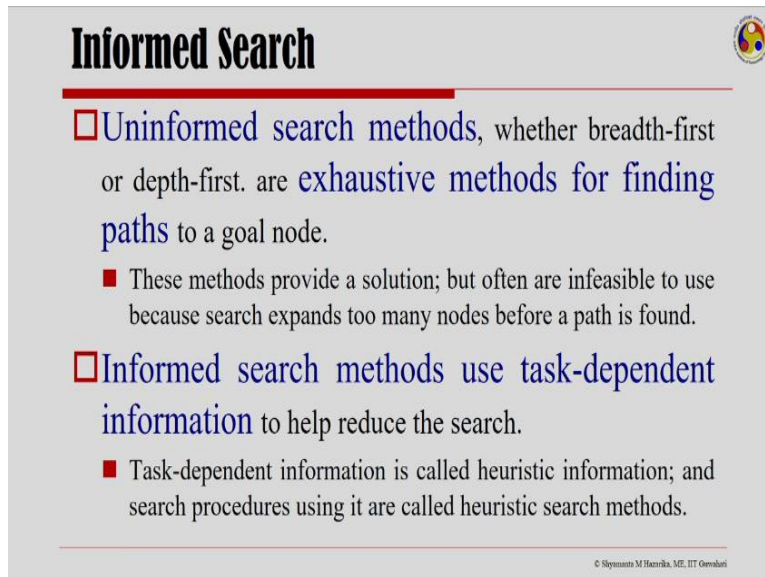
The slide is titled "Informed Search Strategies" and features a list of topics. The title is underlined with a red line. The list includes:

- Heuristic Search
- Types of Heuristic Search
 - Best First Search
 - Algorithm A*
 - Local search
 - Hill Climbing
 - Simulated Annealing

© Shyamanta M Hazarika, ME, IIT Guwahati

We would first look at heuristic search. We would thereafter look at the types of heuristic search. We would typically do 2 interesting types. First, the best-first search. We will discuss algorithm A and thereon define algorithm A *. We would then look at local search and discuss hill climbing and simulated annealing.

(Refer Slide Time: 01:39)



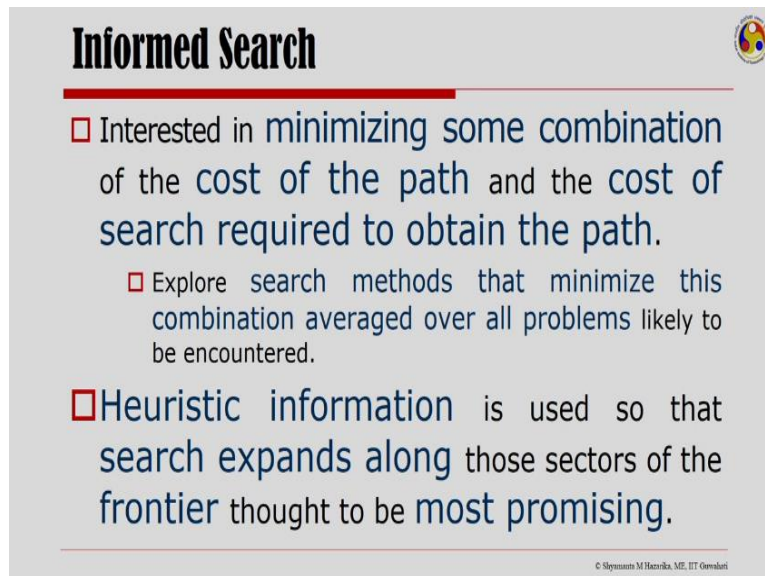
Informed Search

- Uninformed search methods, whether breadth-first or depth-first, are exhaustive methods for finding paths to a goal node.
 - These methods provide a solution; but often are infeasible to use because search expands too many nodes before a path is found.
- Informed search methods use task-dependent information to help reduce the search.
 - Task-dependent information is called heuristic information; and search procedures using it are called heuristic search methods.

© Sreyananti M Hazrika, M.E., IIT Guwahati

Uninformed search methods that we have discussed so far, whether breadth-first or depth-first are exhaustive methods for finding paths to a goal node. These methods provide a solution, but often are infeasible to use, because search expands too many nodes before a path is found. On the other hand, we have informed search methods which use task-dependent information to reduce the search. The task-dependent information is called heuristic information and search procedures that use it are called heuristic search methods.

(Refer Slide Time: 02:29)



Informed Search

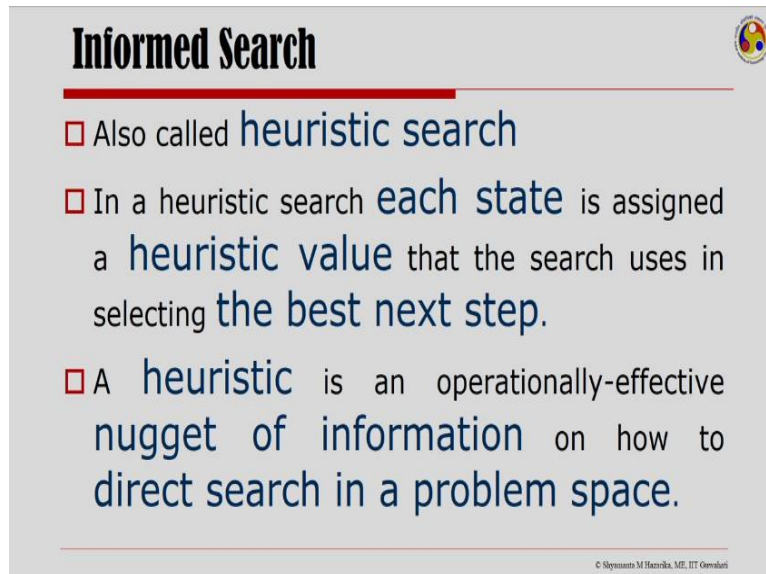
- Interested in minimizing some combination of the cost of the path and the cost of search required to obtain the path.
 - Explore search methods that minimize this combination averaged over all problems likely to be encountered.
- Heuristic information is used so that search expands along those sectors of the frontier thought to be most promising.

© Sreyananti M Hazrika, M.E., IIT Guwahati

We are interested not only in minimizing the cost of the path, but also the cost of the search required to obtain the path. That is, we are interested in minimizing some combination of the two. Search methods that minimize this combination averaged over all populations is what

we are likely to look at today. Heuristic information is used so that search expands along those sectors of the frontier which are thought to be most promising.

(Refer Slide Time: 03:13)



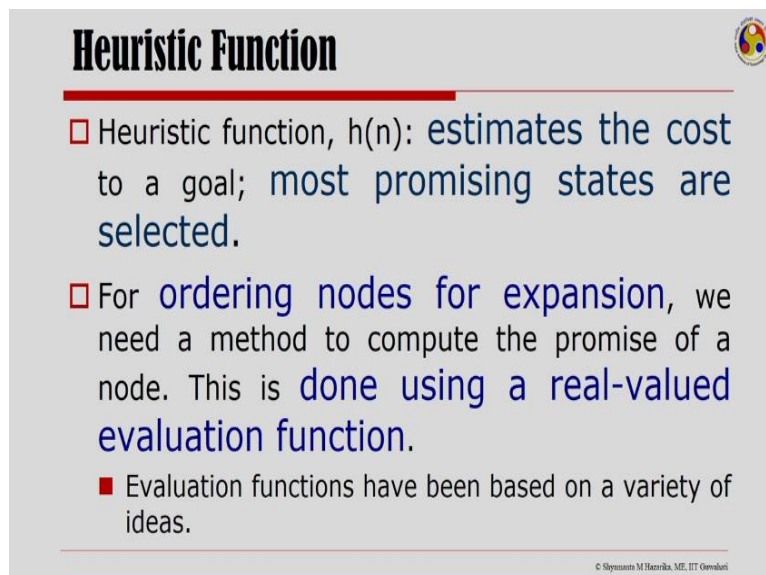
Informed Search

- Also called **heuristic search**
- In a heuristic search **each state** is assigned a **heuristic value** that the search uses in selecting **the best next step**.
- A **heuristic** is an operationally-effective **nugget of information** on how to **direct search in a problem space**.

© Sreyas M. Hatricka, ME, IIT Guwahati

As I told you before, we call them heuristic search, for they use task-dependent information. In a heuristic search, each state is assigned a heuristic value, a value that somehow reflects its importance in the search and is used in the search to select the best next step. A heuristic is an operationally effective nugget of information and directs the search in a problem space.

(Refer Slide Time: 03:54)



Heuristic Function

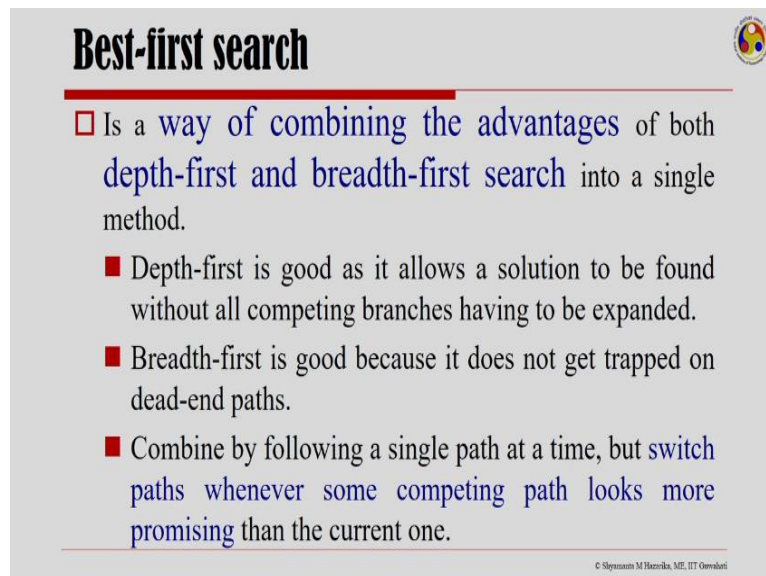
- Heuristic function, $h(n)$: **estimates the cost to a goal; most promising states are selected**.
- For **ordering nodes for expansion**, we need a method to compute the promise of a node. This is **done using a real-valued evaluation function**.
 - Evaluation functions have been based on a variety of ideas.

© Sreyas M. Hatricka, ME, IIT Guwahati

We have looked at heuristic functions in great detail in our last lecture. We continue here with a review. The heuristic function written as h of n , estimates the cost to a goal. The most promising states are selected. For ordering nodes for expansion in our search, we need a method to compute the promise. And this is done using a real-valued evaluation function.

Evaluation functions have been based on a variety of ideas. Like for the 8-puzzle, we had discussed and defined an evaluation function such as counting the number of misplaced tiles.

(Refer Slide Time: 04:52)



Best-first search

- Is a way of combining the advantages of both depth-first and breadth-first search into a single method.
 - Depth-first is good as it allows a solution to be found without all competing branches having to be expanded.
 - Breadth-first is good because it does not get trapped on dead-end paths.
 - Combine by following a single path at a time, but switch paths whenever some competing path looks more promising than the current one.

© Sreyas M. Harekha, ME, IIT Guwahati

The best-first search that we will discuss first in our informed search strategies is a way of combining the advantage of both the depth-first and the breadth-first search into a single method. We need to recall that depth-first is good, as it allows a solution to be found without all competing branches having to be expanded. On the other hand, breadth-first is good, because it does not get trapped on dead-end paths.

We want to combine this in a best-first search and follow a single path at a time, but switch paths whenever some competing path looks more promising than the current one. That is the philosophy of breadth-first. Take the best of the depth and the best of the breadth-first searches and jump from 1 branch of the search to the other branch to get to the best nodes at every point in time.

(Refer Slide Time: 05:54)

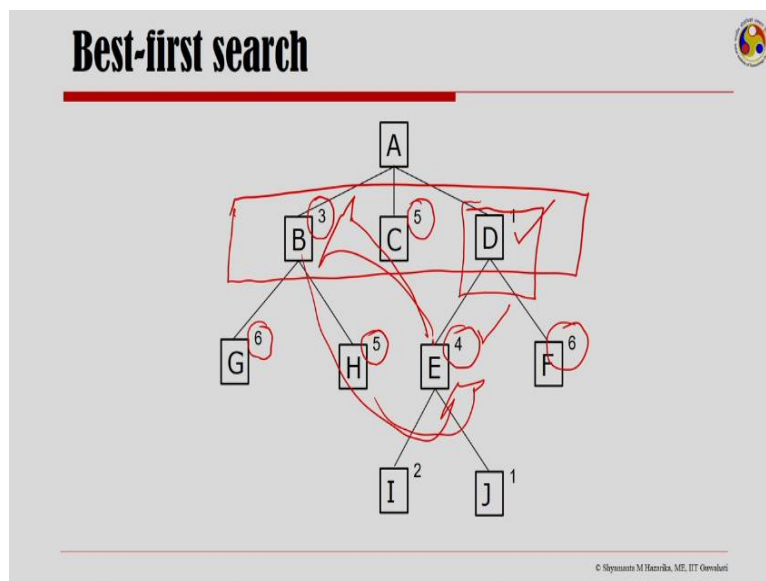
Best-first search

- Idea: use an **evaluation function** $f(n)$ for each node
 - estimate of "desirability"
 - Expand most desirable unexpanded node
- Implementation:
Order the **nodes** in fringe in **decreasing order of desirability**
- Special case:
 - A* search

© Sreyas M Hazare, MIT, IIT Guwahati

The idea is to use an evaluation function for each node that estimates the desirability and then expand the most desirable unexpanded node. We order the nodes in decreasing order of desirability. And one of the special case is the A* search.

(Refer Slide Time: 06:20)



Let us look at an example here. Let us say we have a problem. And where we are searching a tree and A node is expanded to give us node B, node C and node D. Let us now try to compute some heuristic function that gives us a value of these nodes, the heuristic value of these nodes. So, let us say B has a value of 3, C has a value of 5 and D has a value of 1. The next thing that we would love to do if these values reflect somehow the distance to the goal, we would love to expand node B, for it has the minimum of the distance to the goal.

So, we expand D and let us say we end up having 2 nodes E and F. And let us say these heuristic values that I get of all the evaluation function at node E and node F is 4 and 6 respectively. Now, if you compare the heuristic values at B which is still unexpanded at C which is still unexpanded; and the unexpanded nodes E and F, we see that B looks most promising. And therefore, a best-first search would expand B rather than expanding E or F as it would have been done in case of a depth-first search.

Therefore, the next thing that is expanded is B. And let us say we have a node out of B which is G and H. And if we look at their heuristic values, then we have 6 and 5. In that case, interestingly, once again the search would jump back to E. Because, if you see, G has a value of 6, whereas H has a value of 5. And E looks most promising at this point of time. The search would then expand E.

This is the basic idea of a breadth-first search. To once again reiterate what I have just now said. In a breadth-first search, we do go about looking at expanding every node as I did it for the first time, but then, out of this, I decided to go and take down D, not B or C, as D looked most promising and I expanded its nodes. But while doing so, I jumped back to the node which was not expanded and even looked promising.

So, from doing D and not doing E and F, I jumped back from here to expanding C, sorry expanding B. And once B was expanded, I realized that nodes G and H are less promising than E. I come back and expand E. So, this is what a breadth-first search does. It tries to expand the best promising node or while shifting from doing depth-first to breadth-first and breadth-first to depth-first again.

(Refer Slide Time: 10:07)

Graph Search

Procedure **GRAPHSEARCH**

- Create a search graph **G** consisting solely of the start node **s**.
- Put **s** on a list called **OPEN**
- Create a list called **CLOSED** that is initially empty.
- Loop: If **OPEN** is empty, exit with failure
- Select first node on **OPEN**, remove it from **OPEN** and put it on closed. Call this node **n**.
- If **n** is goal node, exit successfully with the solution obtained by tracing a path along the pointers from **n** to **s** in **G**
- Expand node **n** generating the set **M** of its successors and install them as successors of **n** in **G**.
- Establish a pointer to **n** from those members of **M** that were not already in **G**. Add these members of **M** to **OPEN**.
- For each member of **M** already in **G** decide whether or not to redirect its pointer to **n**.
- For each member of **M** already on **CLOSED**, decide for each of its descendants in **G** whether or not to redirect its pointer.
- ✓ Reorder the list **OPEN**, according to heuristic merit.
- Go LOOP.

© Sreyas M Hazrika, ME, IIT Guwahati

So, if you recall, we had introduced this graph search algorithm, where we had said that we will have 2 lists. One called open and the other called closed on which we shall maintain the nodes that are to be expanded and that have already been expanded. So here, continuing that discussion, as I am dealing with A B C D and then expanding E and then going back and expanding B, the only point in which this algorithm of graph search would make a difference is on the reorder of the open list.

When I reorder the open list, I will now order it according to the heuristic merit. And therefore, the node that would be expanded next would be at the front, allowing me to take the most promising node first.

(Refer Slide Time: 11:17)

Algorithm A

- Idea: avoid expanding paths that are already expensive
- Evaluation Function $f(n)$ can be defined so that its value at any node **n** estimates the sum of cost of minimal cost path from start node **s** to node **n** together with the cost of minimal cost path from node **n** to goal node.
- $f(n)$ is an estimate of the cost of a minimal cost path constrained to go through node **n**.

© Sreyas M Hazrika, ME, IIT Guwahati

Algorithm A is about avoiding expanding paths that are already expensive. And here, the evaluation function can be defined so that its value at any node estimates the cost of the minimal cost path from the start node s to the node n . So, that is the minimal cost path. And together with it, the cost of the minimal cost path from the node to the goal. So, algorithm A is about; like if you, I have a start node here and somewhere down the line, down there you have a goal, then the evaluation function that I use for algorithm A is about an estimate of the cost that I need to expand to come to this node.

And it is about the cost of coming from here to that particular node; and then, the cost of going from this node to the goal. And this estimate gives me the value of the evaluation function. Now, if you look at this very closely, you would realize that what I am trying to emphasize here is the point that the evaluation function $f(n)$ is an estimate of the minimal cost path that goes through node n at this point of time.

(Refer Slide Time: 12:45)

Algorithm A

- Let function $k(n_i, n_j)$ give the **actual cost of a minimal cost path** between two arbitrary nodes n_i and n_j . ✓
- Cost of minimal cost path from node n to some particular goal node t_i is then given by $k(n, t_i)$. ✓
- $h^*(n)$ be the minimum of all the $k(n, t_i)$ over the entire set of goal nodes $\{t_i\}$
- $h^*(n)$ is the cost of the **minimal cost path from n to a goal node.** ✓

© Sheykh M. Hossain, M.E., IIT Guwahati

So, in order to get this more correctly, let us introduce some notations. Let us say we have a function k, n sub i and n sub j that gives the actual cost of a minimal cost path between 2 nodes n sub i and n sub j . Now, the cost of minimal cost path from node n to some particular node t sub i is then given as k of n t sub i . $h^* n$ is the minimum of all such paths over the entire set of goal nodes, t sub i that I could generate. And that is the heuristic value. So, $h^* n$ is the cost of the minimal cost path from n to a goal node.

(Refer Slide Time: 13:40)

Algorithm A

- Cost of an optimal path from a given start node, s to some arbitrary node n is $k(s, n)$.
- $g^*(n) = k(s, n)$; for all n accessible from s .
- Define function f^* so that its value $f^*(n)$ at any node n is the actual cost of an optimal path from node s to node n plus the cost of an optimal path from node n to a goal node.

$$f^*(n) = g^*(n) + h^*(n)$$

© Sreyas M Hazrika, M.E., IIT Guwahati

That is the cost of an optimal path from a given start node s to some arbitrary node n . So, that could be written as $k(s, n)$. And we write this as $g^*(n)$. So, $g^*(n)$ is the cost of the optimal path from a start node to a node n . And together with $h^*(n)$, we define $f^*(n)$ which is the actual cost of an optimal path from node s to node n plus the cost of an optimal path from node n to the goal node. So, it has 2 things that I just now told you in the previous slide.

I have this start node s ; and somewhere I have an intermediary node n . And down the line, I have a goal node g . So, I take this as $g^*(n)$, the cost of coming from the start to n , the optimal cost. And I take this, the cost of going from n to the cost of going from n to g which is $h^*(n)$. The evaluation function f^* is defined as $g^*(n) + h^*(n)$.

(Refer Slide Time: 15:11)

Algorithm A

- Evaluation function f is an estimate of f^* . This estimate can be given by

$$f(n) = g(n) + h(n)$$

- $g(n)$ = cost so far to reach n ; is an estimate of g^*
- $h(n)$ = estimated cost from n to goal is an estimate of h^* ; for which we rely on heuristic information.
- $f(n)$ = estimated total cost of path through n to goal

- Graph search algorithm using this evaluation function for ordering nodes in called algorithm A.

© Sreyas M Hazrika, M.E., IIT Guwahati

Now, the evaluation function f is an estimate of f^* . And this estimate can be written as, f is = g of n + h of n , where g of n is the cost so far to reach n and actually is an estimate of g^* . h of n is the estimated cost from n to goal and is an estimate of h^* . And for h , this estimate alone, we rely on heuristic information. Given these 2 estimates, we have f of n which is the estimated total cost of a path through n to the goal. Any graph search algorithm using this evaluation function for ordering nodes is called algorithm A.

(Refer Slide Time: 16:04)

A* Algorithm

- If h is a lower bound on h^* i.e., if $h(n) \leq h^*(n)$ for all nodes n , then algorithm A will find an optimal path to a goal. ✓
- When algorithm A uses a h function that is a lower bound on h^* , we call it algorithm A*. ✓
 - The best-first algorithm that was described is a simplification of algorithm A*.
 - Since $h=0$ is certainly a lower bound on h^* ; BFS finding minimal length paths follows as a special case.

© Sreyas M. Hariharan, ME, IIT Bombay

If now, you take the lower bound of h^* ; that means, when you estimate h , you take the lower bound of the estimate for all nodes. Then algorithm A will find an optimal path to the goal. When algorithm A uses such an heuristic function, that is a lower bound on h^* , we call it algorithm A*. The best-first algorithm that I had little while ago described is a simplification of algorithm A*.

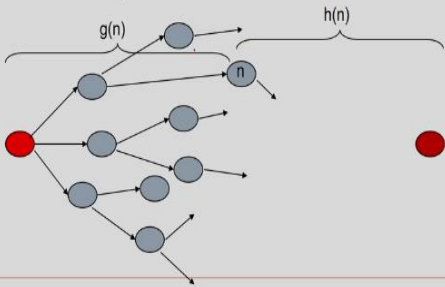
Now, it is interesting to note that if we take $h = 0$, which is certainly a lower bound on h^* , we end up having breadth-first search, finding minimal length paths. The breadth-first search therefore finding minimal length paths follow as a special case of this.

(Refer Slide Time: 16:55)

A* Algorithm

$f(n) = g(n) + h(n)$

- $g(n)$ = cost from the starting node to reach n
- $h(n)$ = estimate of the cost of the cheapest path from n to the goal node



© Shyamant M Hazrika, ME, IIT Guwahati

So, let us try to understand the * algorithm. The evaluation function f of n is written as g of n + h of n , where g of n is the cost from the starting node to reach n . Whereas h of n is the estimate of the cost of this cheapest path from n to the goal. So, here h of n is an estimate, is a heuristic value. So, if you see, we can be very sure of g n , whereas h n is an estimate.

(Refer Slide Time: 17:32)

Solving 8-Puzzle using A*

✓ start

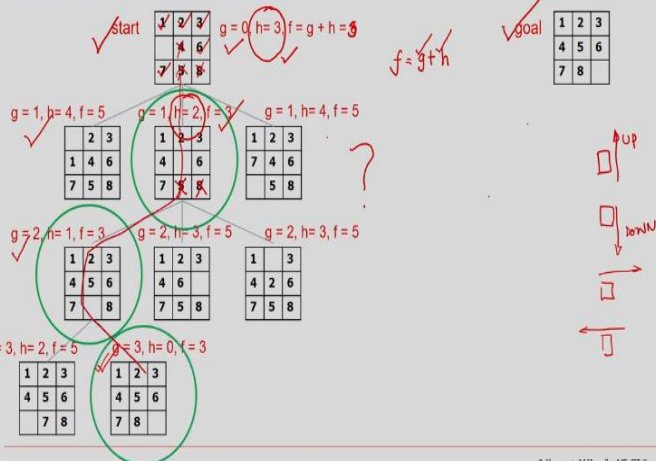
2	3	7
1	4	6
7	5	8

 $g=0, h=3, f=g+h=3$

$f = g + h$

✓ goal

1	2	3
4	5	6
7	8	



© Shyamant M Hazrika, ME, IIT Guwahati

Let us try to understand this by solving our favorite 8-puzzle using the A * algorithm. Here is a start state and here is the end state that I want to reach. Now, just to remind you of the 8-puzzle game that we want to solve here, we want a path from the start to the goal. Now, when we want a path, we are able to operate on this blank space. Given a blank space, we could lead this state to newer states.

So, let us see how many states we can end up doing. So, on moving 1 down, we have one state; on moving 4 out, we have another state; moving 7 up, we have a third state. Now, the question is, given this 3 states, which is the state that I expand next? Recall that the A* algorithm uses an evaluation function which is written as $f = g + h$ at every node n . So, we start computing g , the cost of reaching the n th node; and h the estimate of going from n to the goal node.

Let us do it for each of the nodes that we have expanded. So, at the first node to start with, g is obviously 0. We have not expanded anything to come to this node as of yet. And let us take h , the heuristic which we will try to count the misplaced tiles, the number of misplaced tiles. So, we want to count the number of misplaced tiles as our heuristic function h . So, in here, if you compare it with the goal state, we could see that 1, 2, 3 are in its place.

4 is not in its place. Then 6 is in its place. 7 is in its place. 5 is not in its place and 8 is not in its place. So, the number of misplaced tiles here counts as 3. Given the fact that I have not expanded anything to come to the node and my heuristic value which I have stated as number of misplaced tiles as some heuristic or estimate of going from n to the goal node; I have counted as 3. The evaluation function will have a value of $f = g + h$, which is 3.

I compute the other heuristic values here. So, at this point $g = 1$ and $h = 4$ and I end up having $f = 5$. At this point, I have $g = 1$, $h = 2$, $f = 3$. And at this point, $h = 4$. And $f = 5$. Just for a cross check, I just want to show you one more time, counting of the misplaced tile heuristics. Here, if you look at the number of misplaced tiles; 1, 2, 3 are in its place; 4 is in its place, 6 is in its place, 7 is in its place.

Only tiles which are not in its place is 5 and 8. Therefore, I have an h of 2 and leading to an f of 3, depending on the value of f , I pick up this middle state there. And we expand this node to get 3 newer nodes. And then, compute the heuristic values of each of them. Note that at this point g becomes 2 and I count the number of misplaced tiles to know about h . That leads to $f = 3$, $f = 5$ and $f = 5$ again for the last node here.

Therefore, I decide to expand this node at the start. When I do this, I end up having 2 nodes, either moving 7 to the blank or moving 8 to the blank. And the, one of them is the goal node. That you can still see by computing the heuristic function. And you would have $g = 3$ at this

point of time. You would have 2 misplaced tiles here. And therefore, f is 5. In this node, we have $g = 3$, $h = 0$ and f therefore is 3.

And this is what I would ideally take for expansion. But then, I will realize that that is the goal node and my algorithm length. So, what we have realized is that this A^* algorithm could take me to goal in such a nice directed way.

(Refer Slide Time: 23:18)

Observations

- g lets us choose which node to expand next on the basis not only of how good the node itself looks (as measured by h), but also on the basis of how good the path to the node was!
- The estimator of h^* , distance of a node to the goal; if h is a perfect estimator of h^* , A^* will converge immediately to the goal with no search.
 - Better the estimate h , closer we would be to the direct approach.
 - On the other hand, if h is zero, the search would be controlled by g ; $g = 0$: random and $g = 1$: BFS.

© Sreyanath M. Haririka, ME, IIT Overland

The g that I am using in the A^* algorithm lets us choose which node to expand next on the basis not only of how good the node itself looks as measured by the heuristic function, but also on the basis of how good the path to the node was. Therefore, g carries a lot of weight. The estimator of h^* , the distance of a node to the goal is another interesting observation. If h is a perfect estimator of h^* , A^* will converge immediately to the goal without no search.

Better the estimate of h , closer we would be to the direct approach. On the other hand, if h is 0, the search would be controlled by g . And if g is 0, we would end up having random search. If we consider g at every step to be 1, we would end up having a breadth-first search.

(Refer Slide Time: 24:20)

Admissibility of A*

- The A* algorithm, depending on the heuristic function, finds or not an optimal solution. If the **heuristic function is admissible**, the **optimization is granted**.
- A **heuristic function is admissible** if it satisfies the following property:

$$\forall n \ 0 \leq h(n) \leq h^*(n)$$

h(n) has to be an optimistic estimator; it never has to overestimate h(n).*
- Using an **admissible heuristic function guarantees that a node on the optimal path never seems too bad** and that it is considered at some point in time.

© Shekhar M Hazare, ME, IIT Guwahati

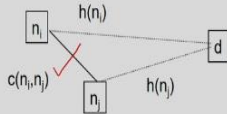
Now, we talk of admissibility of A*. The A* algorithm depending on the heuristic function finds or not an optimal solution. If the heuristic function is admissible, optimization is guaranteed. A heuristic function is admissible if it satisfies the following property. For every n, the h n is bonded by 0 and h* n. That means, h n has to be an optimistic estimator. On the other hand, it never has to overestimate h* n. Using admissible heuristic function guarantees that a node on the optimal path never seems too bad and that it is considered at some point in time.

(Refer Slide Time: 25:10)

Admissibility of A*

- A* generates an **optimal solution** if h(n) is an admissible heuristic and the search space is a tree:
 - h(n) is **admissible** if it never overestimates the cost to reach the destination node
- A* generates an **optimal solution** if h(n) is a consistent heuristic and the search space is a graph:
 - h(n_i) is **consistent** if for every node n_i and for every successor node n_j of n_i:

$$h(n_i) \leq c(n_i, n_j) + h(n_j)$$



© Shekhar M Hazare, ME, IIT Guwahati

A* guarantees an optimal solution if h n is an admissible heuristic and the search space is a tree. Now, h n is admissible as I told you in the previous slide, if it never overestimates the cost to reach the destination node. A* generates an optimal solution if h n is a consistent

heuristic and the search space is a graph. Now, h_n is a consistent heuristic. If for every node, $n_{sub\ i}$ and for every successor node $n_{sub\ j}$ of $n_{sub\ i}$, we have $h_{n_{sub\ i}} \leq c(n_{sub\ i}, n_{sub\ j}) + h_{n_{sub\ j}}$.

This is what I mean that the estimate that I make of reaching n_j , the estimate about n_j to the goal and about n_i to the goal should take into account the cost of coming from n_i to n_j . If that happens, then we have a consistent heuristic.

(Refer Slide Time: 26:19)

Optimality of A*

- A* expands nodes in order of increasing f value; Gradually adds f -contours of nodes.

© Sreyanath M Hazareika, M.E., IIT Guwahati

A* expands nodes in order of increasing f then. And what really happens is that, it gradually adds the f -contours, one after another into its search space. And as it encounters gradually f -contours of nodes, it goes towards more promising nodes.

(Refer Slide Time: 26:47)

Optimality of A*

- Suboptimal goal G_2 has been generated and is in the fringe.
- Let n be an unexpanded node in the fringe such that n is on a shortest path to an optimal goal G .
- If A* is optimal, $f(n) < f(G_2)$
 - $f(G_2) = g(G_2)$ ✓ since $h(G_2) = 0$ ✓
 - $f(G) = g(G)$ ✓ since $h(G) = 0$ ✓
 - $g(G_2) > g(G)$ ✓ since G_2 is suboptimal
 - $f(G_2) > f(G)$ ✓
 - $h(n) \leq h^*(n)$ ✓ since h is admissible
 - $g(n) + h(n) \leq g(n) + h^*(n)$ ✓
 - $f(n) \leq f(G)$ ✓ since $g(n) + h(n) = f(n)$ and $g(n) + h^*(n) = f(G)$
 - $f(n) < f(G_2)$ ✓

© Sreyanath M Hazareika, M.E., IIT Guwahati

We would now look at the optimality of A^* and prove it by an intuitive method. Let us say we have a suboptimal goal G_2 in the fringe and we have a node n which is not yet expanded. Let n be in the fringe as well and let it be on a shortest path to the optimal goal G . Now, A^* is optimal if the heuristic value at n is less than the heuristic value at G , f , the heuristic value at G_2 . Now, we know that the f of G_2 is $= g$ of G_2 because G_2 is a goal and h of G_2 is 0.

Further, G also being a goal, f of G is $= g$ of G , since h of G is 0. Now, we have been told that G_2 is suboptimal. That would mean that the cost of coming to G_2 which is g of G_2 is greater than the cost of coming to G . Now, we know that g of G_2 is f of G_2 . Therefore, f of G_2 is greater than f of G . Now, since we are talking of an admissible heuristic, we know that h of n is less than equal to $h^* n$.

And therefore, from this, we could write that g of $n + h$ of n would be less than equal to g of $n + h^* n$. Now, just to recall that g of $n + h$ of n is nothing but the heuristic function and value of the evaluation function f of n . And g of $n + h^* n$ is f of G . So, we could write here that f of n is less than equal to f of G . And given that f of G_2 is greater than f of G , we can conclude that f of n is less than equal to f of G_2 . So, that is how we could say that A^* is optimal.

Because, any node that it picks up for expansion, the evaluation function at that node is going to be less than a sub optimal goal. Now, we will focus on other interesting portion of today's lecture, which is about local search. So, what are local search algorithms?

(Refer Slide Time: 29:31)

Local search algorithms

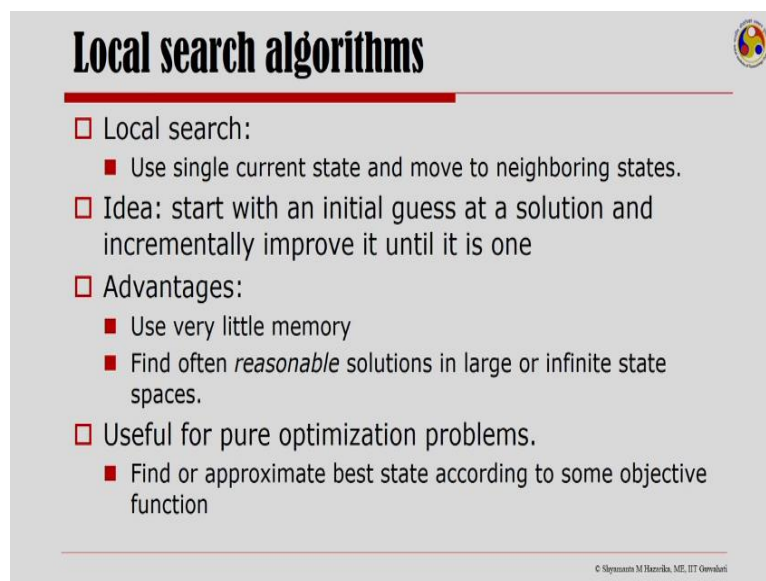
- In many optimization problems, the state space is the space of all possible complete solutions
 - Objective function tells us how "good" a given state is; Find the solution by minimizing or maximizing the value of this function
- These algorithms do **not** systematically explore all the state space.
 - Heuristic (or evaluation) function reduce the search space (not considering states which are not worth being explored).
 - Algorithms do not **usually** keep track of the path traveled.
 - The memory cost is minimal. ✓
 - This total lack of memory can be a problem (i.e., cycles).

© Shyamant M Hazrika, ME, IIT Guwahati

In many optimization problems, the state space is the space of all possible complete solutions. Objective functions tells us how good a given state is. Find the solution by minimizing or maximizing the value of this function. These algorithms do not systematically explore all the state space. Heuristic or evaluation function reduce the search space which are not worth being explored. Algorithms do not usually keep track of the path. The memory cost is minimal therefore.

But one thing to be noted here is, that the total lack of memory at times can be a problem, because we may end up having cycles. Never the less, these are popular search algorithms.

(Refer Slide Time: 30:25)



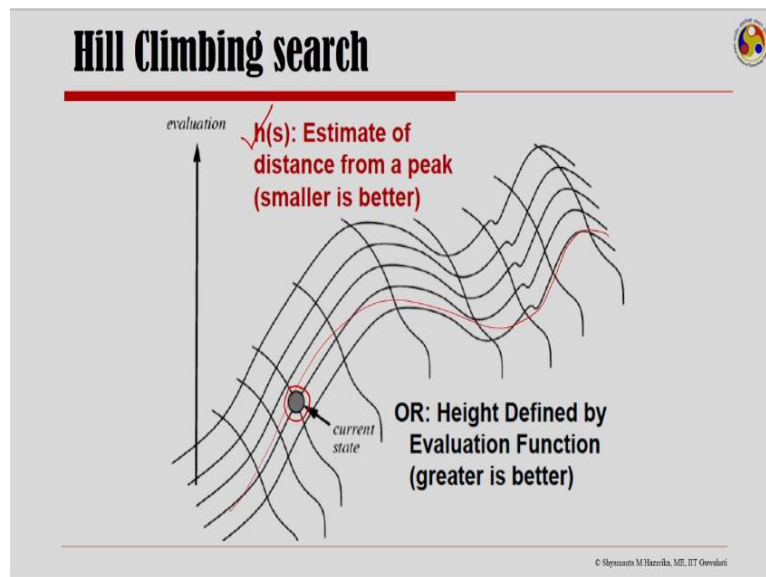
Local search algorithms

- Local search:
 - Use single current state and move to neighboring states.
- Idea: start with an initial guess at a solution and incrementally improve it until it is one
- Advantages:
 - Use very little memory
 - Find often *reasonable* solutions in large or infinite state spaces.
- Useful for pure optimization problems.
 - Find or approximate best state according to some objective function

© Sreyas M Hazare, ME, IIT Guwahati

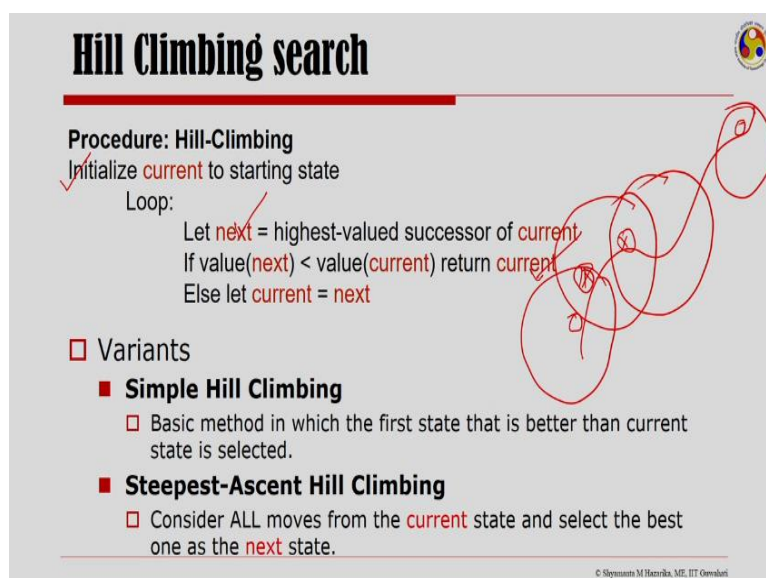
The local search uses single current state and move to neighboring states. The idea is to start with an initial guess at a solution and incrementally improve it until it is one. The advantages of local search is, as I told you, uses very little memory, find often reasonable solutions in large or infinite state spaces. It is useful for pure optimization problems; find or approximate best state according to some objective function.

(Refer Slide Time: 31:07)



We will look at one such local search algorithm called the hill climbing search. The idea is similar to climbing a hill. So, the surface looks like something of a hill with lot of ups and downs. And let us say we are here in our search. We can be using 2 different types of heuristic functions. We may estimate the distance from a peak to arrive at the peak. For such a heuristic estimate, the smaller is the better or we may be using the height defined by the evaluation function. And as you can guess, in a hill climbing search, if I am talking of height, greater is better.

(Refer Slide Time: 31:59)

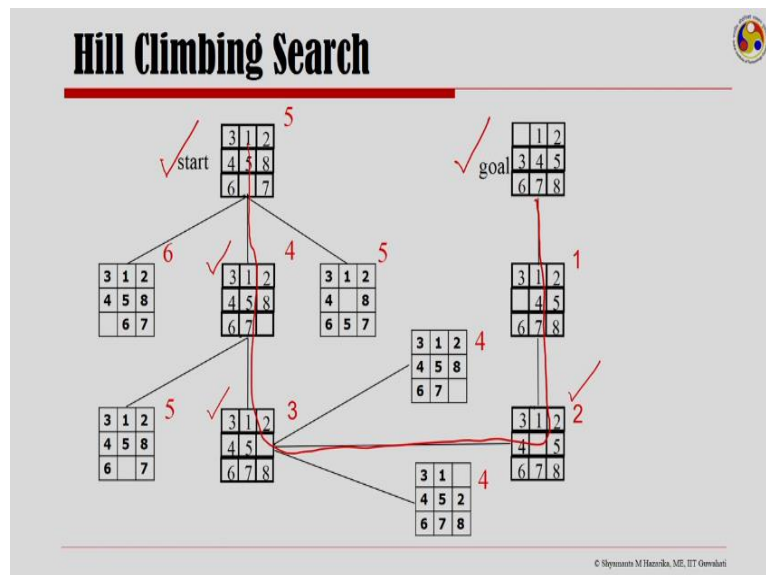


So, the hill climbing procedure, if you allow me to put, is very simple. You just initialize a state called current to the starting state. And then, you initialize a state called next, which takes the highest valued successor of current. If the value of next is value of current the value

of current, we will return current; else, we will let current equal to next. And this continues. So basically, if I am looking for reaching this point from here, I will look at my neighborhood and move to this point, which is possibly the best.

And then, make this my next point and increase my search space. So, pick up this, move here, make my search space here. And I will slowly but surely x reaching the height. But, is it that simple? We will look at it in a minute. There are variants to the hill climbing algorithm. The simple hill climbing is a basic method in which the first state that is better than the current state is selected, as I was trying to explain this. There is other called the steepest-ascent hill climbing where we consider all moves from the current state and select the best one as the next thing.

(Refer Slide Time: 33:25)

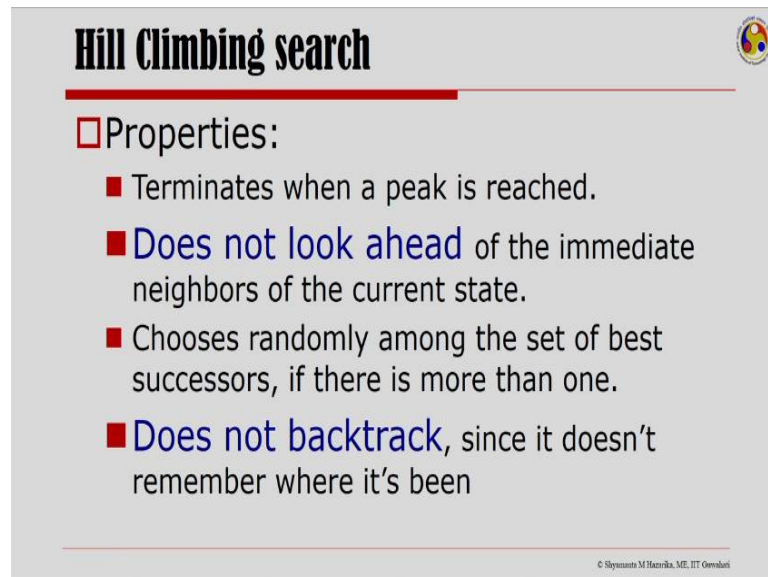


So, let us take our 8-puzzle game once more and see and understand hill climbing. Let us take this to be our start state and here is our goal state. We will use again our heuristic function of the number of misplaced tiles. So, we will try to expand this node and get the newer nodes, but then compute the number of misplaced tiles for each one of these nodes. So, this has 5 misplaced tiles; this has 6; this has 4 and this has 5.

So, I would love to expand the one with the lowest number of misplaced tiles. I expand that node and end up having 2 more nodes here. And then, we compute the heuristic values 5 and 3. Therefore, we opt to expand this node. This node is expanded and ends up in 3 more nodes. We compute the heuristic values. This node with the lowest heuristic value is expanded next. We end up having this node and then that is expanded to reach to the goal.

So, if you give some thought to the whole process, this search that I was trying to do was about moving in a way in which the distance to the goal node was decreasing. And I did not take any other path. So, that is the hill climbing search.

(Refer Slide Time: 35:04)



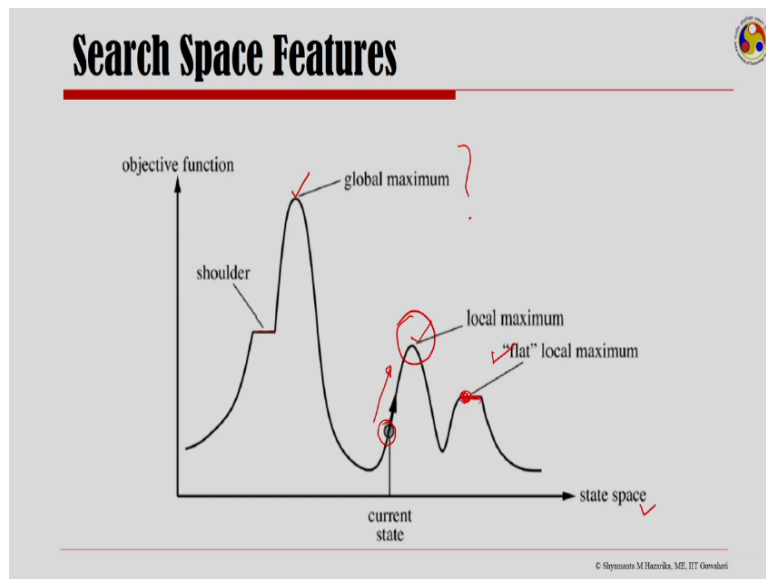
Hill Climbing search

- Properties:
 - Terminates when a peak is reached.
 - **Does not look ahead** of the immediate neighbors of the current state.
 - Chooses randomly among the set of best successors, if there is more than one.
 - **Does not backtrack**, since it doesn't remember where it's been

© Shyamant M Hazra, M.T. IIT Ooverhali

The properties of the hill climbing search is that it terminates when a peak is reached; does not look ahead of the immediate neighbors on the current state and this is what is important. Every time I was expanding the next level, I was only concerned at what are its successors and did not use any estimate of some measure of how far I am from the goal. It chooses randomly among the set of best successors if there is more than one and does not backtrack. Since, if you remember, it does not remember where it has been. It does not keep a track of which state it came from.

(Refer Slide Time: 35:48)



So, this is good if your search space is really looking like a hill. But then, it is not that. Search space features could be very complex as shown in this figure. I have an objective function and the state space. And the state space versus the objective function curve is with many interesting properties. Here is the global maximum. Apart from the global maximum, we have local maximums.

And then, more interestingly, we have flat local maximums as well. Under such a state, if you are searching for just moving towards maximizing your return on the claim and you are somewhere here in the current state and your movement is towards this side rather than this, one would end up having this as the maximum. And we will miss the global maximum. This is a very, very serious problem with hill climbing.

(Refer Slide Time: 36:58)

Drawbacks of Hill Climbing

- **Local Maxima:** peaks that aren't the highest point in the space
 - State that is better than all its neighbours but is not better than some other states farther away.
- **Plateaus:** the space has a broad flat region that gives the search algorithm no direction (random walk)
 - Whole set of neighbouring states have the same value! Not possible to determine the best direction.
- **Ridges:** dropoffs to the sides; steps to the North, East, South and West may go down, but a step to the NW may go up.
 - Orientation of the high region, compared to the set of available moves and directions in which they move, make it impossible to traverse ridges.

© Sreyasath M Hazrati, ME, IIT Guwahati

So, drawbacks of hill climbing is that, peaks that are in the highest point in the local space, the local maxima, we tend to get stuck there. States that is better than all its neighbors, but is not better than some of the other states farther away, would be one with the stuck end. Plateaus: the spaces that has a broad flat region that gives the search algorithm no direction. The whole set of neighboring states that have the same value not possible to determine the best direction.

Like, if you go back and look at that slide, here was a flat region completely and here was another flat region. Now, if I am somewhere here or I do not know which state side to go to when I am on a flat region like this. So, this is another drawback. And then, there are drops off to the sides. Steps to the north, east, south and west may go down. But a step to one of the combinations may go up.

So, orientation of the high region, compared to set of available moves and direction in which they move, make it impossible to traverse ridges. So, with this drawbacks, it is better that we now focus on one of the methods to overcome these. And that method is simulated annealing.

(Refer Slide Time: 38:23)

Simulated Annealing

- A hill-climbing algorithm that **never makes a downhill** move is **guaranteed to be incomplete**, because it can get stuck in a local maximum.
- In contrast, a **purely random walk** - that is, moving to a successor chosen uniformly at random from the set of successors is **complete but extremely inefficient**.
- Combine **hill-climbing with a random walk** in some way to get both efficiency and completeness. **Simulated annealing** is one such algorithm.

© Sreyas M Hazrika, ME, IIT Guwahati

Simulated annealing, the premise of it, the motivation of having it is because hill climbing algorithm that never makes a downhill is guaranteed to be incomplete. What it means is, if I keep on thinking of only climbing up and never think of climbing down again, then I will miss this global maximum. So, what can be done is that, we can think of random walks. But then, purely random walk, that is moving towards successor chosen uniformly at random from the set of successors may guarantee completeness, but is extremely inefficient.

So, we want to do better than this; neither remain incomplete nor wanted to have extreme inefficiencies. So, we combine hill climbing with a random one, in some way to get both efficiency and completeness. Simulated annealing is one such algorithm.

(Refer Slide Time: 39:30)

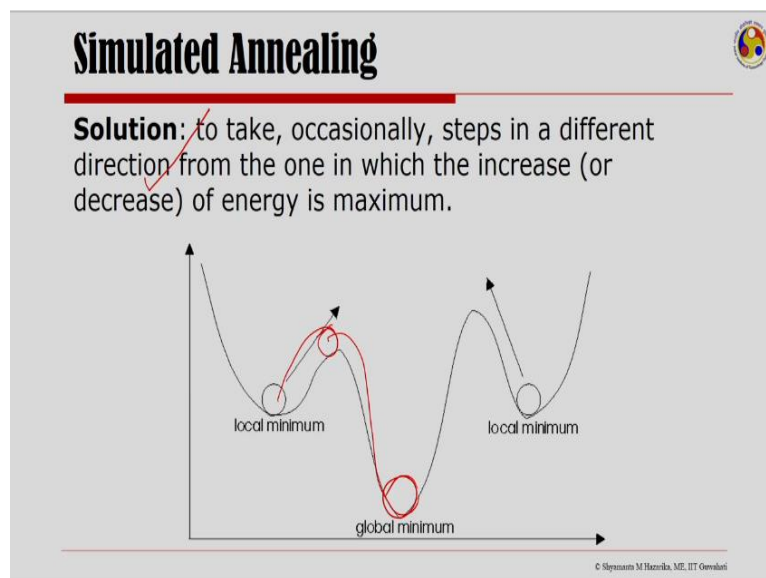
Simulated Annealing

Aim: to avoid local optima, which represent a problem in hill climbing.

© Sreyas M Hazrika, ME, IIT Guwahati

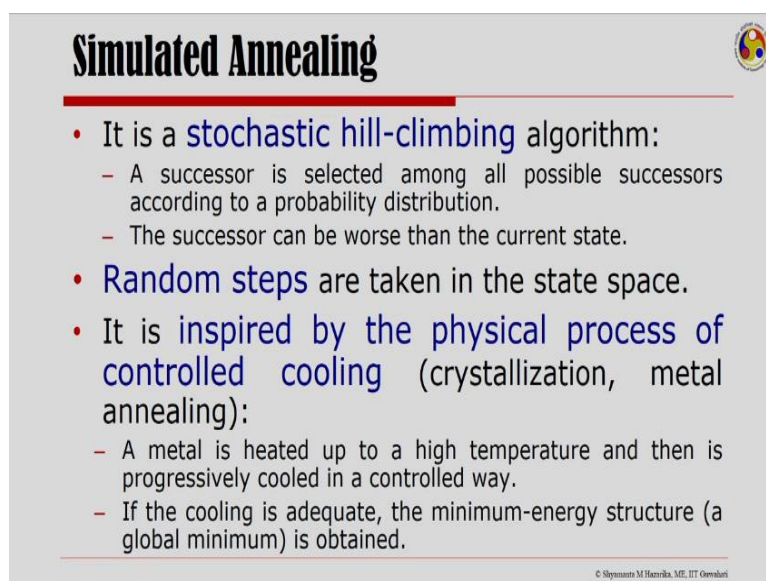
Intuitively, what happens in simulated annealing; let me try to explain to you with this curve. So, here is the state space and the evaluation function. Now, if you see if I, my starting point is here and I was coming down this path. And if only I was looking at the evaluation function. Then, after this point, I will have no more motion taking me out of this. And I will miss this global minimum. So, what can be done at this point of time is that:

(Refer Slide Time: 40:07)



I will occasionally take steps in different direction from which the increase or the decrease, as the case may be, is maximum. So, if I am here with some probability, I may be allowed to come up. And then, there is the chance that I may end up going down this path and arriving at the global minimum. So, this is what is done in simulated annealing.

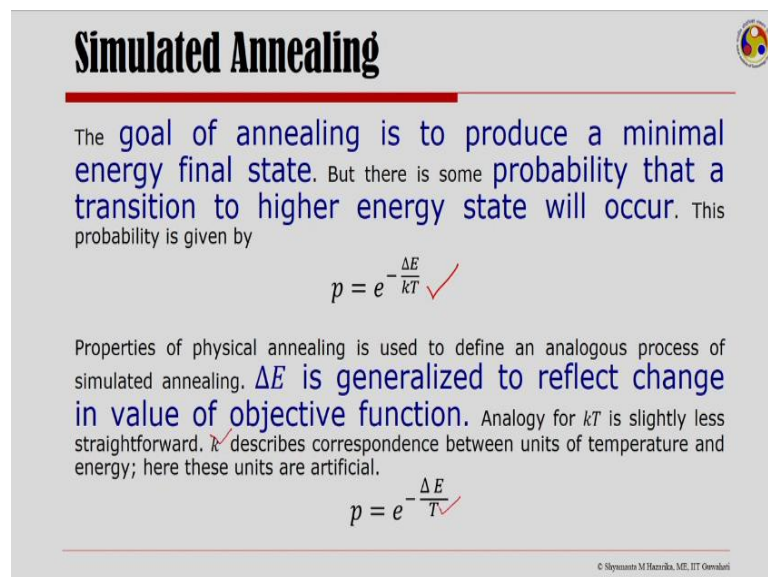
(Refer Slide Time: 40:35)



It is a stochastic hill climbing algorithm. So, a successor is selected among all possible successors, according to some probability distribution. Now, the successor can be worse than the current one. But that is what I want. I do not want to monotonically only go towards 1 path and get stuck in a local maximum or local minimum. Random steps are taken in the state space. And it is interesting to know that this point, that this is inspired by the physical process of controlled cooling, crystallization or metal annealing.

And that is why it is called simulated annealing. Metal annealing or crystallization: a metal is heated up to a high temperature and then is progressively cooled in a controlled way. If the cooling is adequate, the minimum energy structure a global minimum is obtained. This is what we want to achieve in simulated annealing.

(Refer Slide Time: 41:40)



Simulated Annealing

The goal of annealing is to produce a minimal energy final state. But there is some probability that a transition to higher energy state will occur. This probability is given by

$$p = e^{-\frac{\Delta E}{kT}} \checkmark$$

Properties of physical annealing is used to define an analogous process of simulated annealing. ΔE is generalized to reflect change in value of objective function. Analogy for kT is slightly less straightforward. k describes correspondence between units of temperature and energy; here these units are artificial.

$$p = e^{-\frac{\Delta E}{T}} \checkmark$$

© Shyamantak M. Hazra, M.E., IIT Guwahati

The goal of annealing therefore, is to produce a minimum energy final state. But there is some probability that a transition to higher energy state will occur. This probability is given by this function here. Now, properties of physical annealing is used to define an analogous process of simulated annealing. Delta E here, the energy function, is generalized to reflect change in the value of the objective function.

Analogy for kT is slightly different than the physical one. In the physical sense, k describes correspondence between the units of temperature and the units of energy. Here, these units are artificial. And therefore, the k is factored into T . And we use a more simplified equation like this.

(Refer Slide Time: 42:38)

Simulated Annealing

- Best way to **select an annealing schedule**
 - Try **several schedules and observing the effect** on both the quality of the solution that is found and the rate at which the process converges.
- It is **important to realize** that
 - As T approaches zero, the probability of accepting a move to worse goes to zero and simulated annealing becomes identical to simple hill climbing.
 - What matters in computing the probability of accepting a move is the ratio $\frac{\Delta E}{T}$. Thus it is important that **value of T is scaled so that this ratio is meaningful.**

© Shyamant M Hazrika, ME, IIT Guwahati

The best way to select an annealing schedule or change in temperature as we want to do this, is to try several schedules and observing the effect on both the quality of the solution that is found and the rate at which the process converges. It is important to realize 2 things. As T approaches 0, the probability of accepting a move to worse goes to 0 and simulated annealing becomes identical to simple hill climbing.

What matters in computing the probability of accepting a move is the ratio of the energy to the temperature. Thus, it is important that the value of the temperature is scaled, so that this ratio is meaningful. And this is something that one needs to really work on.

(Refer Slide Time: 43:35)

Simulated Annealing

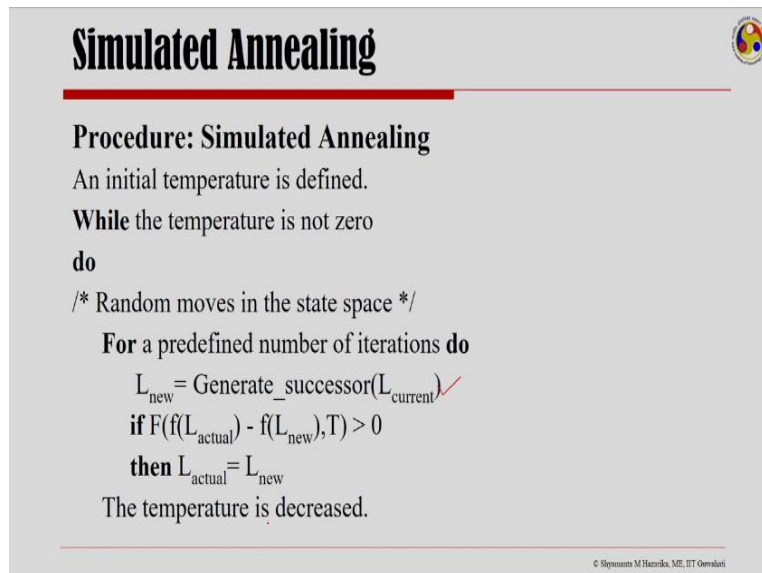
- Terminology from the physical problem is often used.
 - **Temperature** (T) is a control parameter ✓
 - **Energy** (f or E) is a heuristic function about the quality of a state.
- A function F decides about the **selection of a successor** state and depends on T and the difference between the quality of the nodes (Δf): the lower the temperature, the lower the probability of selecting worse successors.
- The *cooling strategy* determines:
 - maximum number of iterations in the search process ✓
 - temperature-decrease steps ✓
 - number of iterations for each step

© Shyamant M Hazrika, ME, IIT Guwahati

Terminology from the physical problem is used here as you have noticed. The temperature T is the control parameter. The energy E or often written also as f is the heuristic function about the quality of a state. The function f decides about the selection of the successor state and depends on the temperature. And the difference between the quality of the nodes; that is, something that I select at one temperature.

What is evaluation function? minus What is the value of the evaluation function at a new temperature? The lower the probability of selecting worse successors. The cooling strategy determines the maximum number of iterations in the search process. The temperature decrease steps and the number of iterations for each step.

(Refer Slide Time: 44:34)



Simulated Annealing

Procedure: Simulated Annealing

An initial temperature is defined.

While the temperature is not zero

do

/* Random moves in the state space */

For a predefined number of iterations **do**

$L_{\text{new}} = \text{Generate_successor}(L_{\text{current}})$

if $F(f(L_{\text{actual}}) - f(L_{\text{new}}), T) > 0$

then $L_{\text{actual}} = L_{\text{new}}$

The temperature is decreased.

© Shyamanta M Hazra, ME, IIT Guwahati

So, here is the simulated annealing procedure. While the temperature is not 0, I would love to do a predefined number of iterations. Okay. I generate the successor of the current successor and look at their f values. Okay? Find that it is still greater than 0. Then I take the actual to be the new one. And the temperature is decreased and this process is continued further.

(Refer Slide Time: 45:02)

Simulated Annealing

- **Main idea:**
 - Steps taken in random directions do not decrease (but actually increase) the ability of finding a global optimum.
- **Disadvantage:**
 - The very structure of the algorithm increases the execution time.
- **Advantage:**
 - The random steps possibly allow to avoid small "hills".
- **Temperature:**
 - It determines (through a probability function) the amplitude of the steps, long at the beginning, and then shorter and shorter.
- **Annealing:**
 - When the amplitude of the random step is sufficiently small not to allow to descend the hill under consideration, the result of the algorithm is said to be *annealed*.

© Shyamant M Hazrika, ME, IIT Guwahati

So, the main idea is that I take steps in random direction. But step taken in random direction do not decrease the ability of finding a global optimum. The disadvantage is: the very structure of the algorithm increases the execution time. However, the random steps possibly allow to avoid small hills; and that is a great advantage. The temperature: it determines through a probability function, the amplitude of the steps, long at the beginning and then shorter and shorter.

An annealing here is when the amplitude of the random step is sufficiently small, not to allow to descend the hill. The result of the algorithm is said to be annealed.

(Refer Slide Time: 45:50)

Simulated Annealing

- It is suitable for problems in which the global optimum is surrounded by many local optima.
- It is suitable for problems in which it is difficult to find a good heuristic function.
- Determining the values of the parameters can be a quite difficult and often requires experimentation.

© Shyamant M Hazrika, ME, IIT Guwahati

So, it is suitable for problems in which the global optima is surrounded by many local optima. It is useful for problems in which it is difficult to find a good heuristic function. Determining the values of the parameters can be quite difficult and often require experimentation. We have now covered the examples that we wanted to talk of in informed search. In our next lecture, we would be covering constraint satisfaction problems. Thank you very much.