**Modified**

Hello everyone and welcome back to Intelligent Systems. Last week in the lecture I gave you an overview about intelligent systems and some related concepts such as artificial intelligence, AI and intelligent agents. IA.

Now, one specific kind of intelligent agents is goal-based agents. And with Go based agents, we also refer to them as problem-solving agents because their task is to have the problem of reaching the Go state from the current state. And they try to solve that problem by finding the solution, finding the path to get to that state, or finding that particular configurations of the state of the world that satisfy that goal.

There are two main techniques to implement problem-solving agents. The first one is search algorithms, and the second one is formulated under the constraint satisfaction problems with constraint solvers algorithms, helping you to solve these constraint satisfaction problems with the search techniques. It is covered very comprehensively in another unit, as Windburn cos three one nine, Introduction to artificial intelligence.

And so in today's lecture, I'm not going to go too deep into search techniques, but the main purpose for today's lecture is on constraint certification problem on so-called CSP. So the overview of today's lecture is I'm going to discuss the problem-solving agents, which is a kind of Go-based agent. And we are talking about how to formulate a problem so that the AI system can solve the problem and find a solution. And I give you some examples. I talk about basic search algorithms and then I will be talking about CSP, which are constraint certification problems.

So some examples for the problems that the AI system can solve, this is the Apuzzo problem. So the apostle having nine cells. One of them is an empty cell, and then the other eight is labeled with the towel from one to eight. And then they will have some initial configuration in this task state. And the problem is to move the towels, the number tout with only one blank cell here in such a way that after you make a number of moves and then the board will reach this ghost plate. And so the towel will be organized in this way. So if you can find a way to move the towels and then get into this Go state, then you solve the problem.

So there is one example of problem-solving and with AI algorithms we will be able to solve this kind of problem. Another very relevant problem is so-called robot navigation. So just imagine that you are making a robot in order to help people. Let's say that in a nursery to help elderly people, because some of them have mobility difficulty. And so the robot needs to help them to move from place to place and then assist them with their daily activities.

For the robot to do that, they have to navigate within an environment. Let's say that this is the layout of the area within the nursery with the gray cells essentially denoting the world that the robot cannot step into. And the plant cell is essentially the space where the robot can go into. The robot currently at the start state in this red cell here. And its goal is to reach this Go state in this green cell here. For instance, it needs to go to this room, this square, in order to assist the elderly person.

So with this one, one way to formulate a problem is to denote the coordinates of different cells within the environment. And so, for instance, the star state will have the coordinates of the S coordinate will be zero and the y-coordinate will be three. And so the star state is a pair of integer, in this case, is three. Then Go state can be represented similarly. And in this particular case, Go state green cells is added coordinate with the S-coordinate being two. And so the Go state is a pair of integer CIS two.

And so you can see that with this kind of problem, the agent, in this case, the robot is trying to solve the problem by finding the path to get the robot from the star state three to the Go state sys two. And again, such algorithms allow you to solve these kinds of problems. Clearly, this environment is just one of the example environments. So the layout of the area can be much bigger, maybe 1000 by 1000 and there may be hundreds of words and so on and so forth.

So those are the two examples of the kind of problems that we hope to use some search algorithm to solve. And for our system to solve those problems with an algorithm, we need to formulate the problems. We will focus on the so-called single-state problems because they are the easiest kind of problems. Imagine that if the robot's information is not complete, it doesn't really know about the layout of the areas. It can only say that the current state can be one of the three possible locations in the area. Then that becomes much harder.

So that is the multi-state problem and we are not dealing with those difficult problems. So we are dealing in this lecture with the simplest kind of problem, the single-state problem. And so we will formulate this kind of problem. Now, the problem is defined by four items. So it's consisting of four components.

The initial state, for instance, in the problem for robot navigation, the initial state is the star coordinate, which is three. Then the agent is equipped with a number of actions it can perform, in particular in that robot navigation problem. Then clearly the robot can go up, go down, go to the left, go to the right. And so those are the possible actions that the robot can perform.

Now, even though there are four actions that the robot can perform, but because the robot may be next to a wall or next to the boundary of the environment at the corner. And so not all actions can be performed. And so the successor function SX allows you to map from a current state x and tell you what are the legitimate actions that the robot can perform.

For instance, in the square three, the robot cannot go to the left because it's already at the boundary of the environment, it could only go up, go down and go to the right. And so in this case, the successor functions will say that in the state three, the successor functions define these three pairs. Action up. And then the robot will be able to go from zero three to or the second pair is performing the action down. The robot is able to go from three to four. Or the robot can also choose to go right, performing the action right. And then the robot will go from the state zero three to state 13.

So the actions that the robot can perform, the legitimate actions that the robot can perform will be represented by the successor function SX for every state x in the environment. So that's the second component. So the first component, the initial state, the second component, successor function, the third component is the Go test.

So in some problems the Go test can be explicit. For instance, in the robot navigation problems, the Go test is essentially saying that the agent has to be reaching the coordinate sys two, that is the Go state. But in some problems the Go test could be implicit. For instance, if the agent's task is to play chess, then when playing chess, then the agent is winning achieving its goal, it is to put its opponent in a checkmate position.

So the goal is satisfying that the position of the chessboard x is a checkmate against the opponent. So the Go test defines the third component and the last component essentially says that there are many ways to go from the initial state to a Go state and some ways could be better, could be more optimal.

And so in order to represent whether one way is better than another, we define the path cost. So getting from the initial state to a Go state requires, you require the action to make many steps. So that sequence of steps is called a path, getting the agent from one state to another state. And in order to show that one path is better than another, we review the cost of the path.

And so we assume in the single-state problem formulation that the path cost is additive. So that means that if you have each step costs a certain cost, then you just sum all the steps and then the cost of all the steps and then you get the cost of the path. And so the assumption is always that the step cost for each step in the past is definite and assumed to be non-negative.

Because if you can take an action. And then instead of its cost, the agent actually benefits the agent, then that is not considered to be a cost and that would make the problem more difficult to solve. And so we want to have additive and non-negative cost. So if you have a problem like this one, then solving the problem essentially means that you produce a solution, which is a sequence of actions leading from the initial state to a goal state.

And remember that there may be many possible goal states. For instance, in playing chess, you can have a Go when you still have ten pieces and the opponents may have eleven pieces, but you put the opponent in a checkmate. In other situations, when you only have five pieces, the opponents have three pieces, and then you put the opponent in a checkmate, you also reach the Go. And so a Go state doesn't have to be unique.

In the case of the example for robot navigation, you can see that there is only one Go state, CIS two. But in many situations the Go state can be multiple and the job is to find one solution that leads to one Go state. All right? So that is the problem. Now we have been able to formulate it and then put all of the components of the problems, like the initial state, like the successor function, like the Go test and the path cost, into some data structure.

And with this data structure, hopefully we are able to have some good algorithms to solve these problems. So the algorithms are put under the so-called research algorithms. And so with these research algorithms, there are many of them, not just a single one, but they follow the same structure by using the tree data structure in order to execute the search systematically.

So let's go back to the robot navigation problem. So the tree search starts with the root node of the search tree. And this root node of the search tree will store the initial state. In this case, that is the coordinate three. Now remember that the successor functions from the state three have three legitimate actions. The agent can go up, go down, or go right and cannot go left because it's already at the boundary of the environment and so cannot go left.

And so with this initial state now expanding this search from this root node of the search, then the system will be able to expand the search and get these three children of this zero three. So if it goes up, it can reach the stage two, if it goes down, it can reach the stage four, and if it goes right, it can reach the stage 13.

Now you can see that the tree has an upside-down structure. So the root node of the tree is actually sitting on the top, and then we expand it downward. Clearly you can also draw the tree with the root node at the bottom and then expand it upward but then that is just the convention. In this case, we choose the root node at the top and expand the tree downward.

And the second thing that you will observe is that after a node has been expanded we will shade it with green color while the nodes that haven't been expanded, they will have the black boundary with white inside. And so these nodes are the candidates for expansion. The question is which node should we expand next?

So there are three, should we expand zero two or four or one three? And as you will see, choosing the node to expand next is actually defining the search strategies. And so let's say that we will choose the leftmost nodes to expand and so we choose zero two then the next one is that we will see that from zero two there are two nodes that we can reach.

So on the left is the boundary of the environment, on the right is a wall. And therefore, from zero two, you can only go up or go down. Go up, get you to one, go down, get you back to three. And so you can see that and after two has been expanded, then you can see that we again shade it with green color. And then the two new children of this node will now be added. One because you can go up from two to one and three because you can go down from zero two back to three.

And now again so the list of nodes that are candidates for expansions are these four, this one, this one, this one and this one. Then again the search algorithm faced the question of these are the four candidates for expansions, which one should it choose to expand next? And that is the essence of the search algorithms. A particular search method will have a way to choose which node to expand next.

Now let's look into the data structure that we will use in order to represent the search algorithm and then that will allow us to implement the search algorithm as well. So here you can see that on the one side you have the state of the problem. So this is the state for instance if we are talking about the aperture problem so this is a state, every configuration, every representation of a physical configuration of the problem is a state.

From one state you can take an action and then you move to another state. On the other hand, the search tree consisting of the search node like this one and the search node itself is a data structure. This data structure contains a number of components. Firstly it says that which state in the state space this node is associated with.

Secondly, the node data structure will also tell you which is the parent node that from this parent node and then expand the parent node and then you obtain this particular node and then it also says that from the parent node which action has been taken in order to bring you to this current node. So for instance, you go right and so on and so forth. And what is the depth of this node within the search tree?

So let's say that if you look into this search tree with these four nodes, so this is a search tree. And let's say that this is the root node of the search tree. Then always the root node has the depth zero. And then clearly this node will have the depth one. This node, we have the depth two. And this node, we have the depth two as well.

In this particular case, this is not the root node. And so there will be some root node up here. And the data structure tell us that the depth of this node is two. And then let's say that every step that you need to take costs one. Then clearly with the depth two, then you go from the root node to this node, you take two steps and that will cost you a path cost of two.

Now with this kind of data structure, you can clearly see that you can navigate back from this particular node to the parent node. From this node, you can also tell from which action it has been generated, and then the parent node will give you the previous node from which it is generated and then the action taken from that parent node. And so that will give you the whole path that goes from the initial state to the current state.

And then you can trace back this path in reverse and that will give you one solution to the problem. If you have another solution then that will require another search, and then that will be different. And so you can have many different solutions to the problem, each solution, the sequence of actions that correspond to the path.

Now having this data structure will allow you to also know how to choose the nodes to expand next. For instance, in the breadth-first search algorithm, you will always choose the nodes that are the shallowest. Because when you go to the shallowest nodes, and it will allow you to reach the goal state more quickly because you always expand nodes at the shallowest.

In the case of the best-first search, then you will choose the nodes that are closest to the goal state. In the case of the depth-first search algorithm, you will always choose the node that is at the deepest possible level in the tree. In the case of the uniform-cost search algorithm, then you will choose the node that has the smallest path cost.

Now, these are the search strategies. And so the search algorithms can be classified into those four categories. Clearly, we will also have some strategies that are intermediate between these four search strategies. And for instance, we can have strategies that allow you to combine depth-first search with best-first search, and that will give you an algorithm called iterative deepening search.

And so the list of search algorithms is actually quite long. And depending on the problem, depending on the strategy, and depending on the characteristics of the search tree, you will have different algorithms that will be suitable to solve the problem

**Summarise**

In this lecture, the focus was on problem-solving agents, specifically goal-based agents, also known as problem-solving agents. These agents aim to reach a goal state from their current state by finding a solution or a path to achieve the desired state. Two main techniques to implement problem-solving agents were discussed: search algorithms and constraint satisfaction problem (CSP) algorithms.

Two example problems were highlighted to demonstrate the kind of problems AI systems can solve using search algorithms. The "puzzle" problem involves arranging numbered tiles in a specific order, and robot navigation entails guiding a robot through an environment to reach a designated location.

The process of formulating a problem for AI systems to solve involves defining four components: the initial state, possible actions, the goal test, and path cost. These components are organized into a data structure, forming the basis for various search algorithms.

The lecture also explored the concept of search trees. Starting from the initial state, nodes are expanded based on the available actions, resulting in a growing search tree. The choice of which nodes to expand next depends on the search strategy being employed, such as breadth-first search, depth-first search, best-first search, and more.

It was highlighted that search algorithms systematically explore the state space until a goal state is reached. Once the goal is achieved, the sequence of actions leading from the initial state to the goal state can be traced back through the search tree. Different search strategies suit different problems and scenarios, offering a range of algorithms to choose from.

***Important***

**INTELLIGENT SYSTEMS**

In today's lecture, I'm going to discuss the problem solving agents, which is a kind of Go based agent. We are talking about how to formulate a problem so that the AI system can solve the problem and find a solution. The main purpose for today lecture is on constraint certification problem on so called CSP.

**TREE SEARCH ALGORITHMS, THE FOUNDATIONS**

The algorithms are put under the so called research algorithms. They follow the same structure by using the tree data structure in order to execute the search systematically. A particular search method will have a way to choose which node to expand next.

**EXPLORING A SEARCH TREE WITH THE ALGORITHMS**

The search tree consists of the search node like this one. The search node itself is a data structure. From one state you can take an actions and then you move to another state. And then the path cost that form the initial state gets in through that path.

**Original**

Hello everyone and welcome back to Intelligent Systems. Last week in the lecture I gave you an overview about intelligent systems and some related concepts such as artificial intelligence, AI and intelligent agents. IA. Now, one specific kind of intelligent agents is goal based agents. And with Go based agents, we also refer to them as problem solving agents because their task is to have the problem of reaching the Go state from the current state. And they try to solve that problem by finding the solution, finding the path to get to that state, or finding that particular configurations of the state of the world that satisfy that goal. There are two main techniques to implement problem solving agents. The first one is search algorithms, and the second one is formulated under the constraint satisfaction problems with constraint solvers algorithms, helping you to solve these constraint satisfaction problems with the search techniques. It is covered very comprehensively in another unit. As Windburn cos three one nine. Introduction to artificial intelligence. And so in today's lecture, I'm not going to go too deep into search techniques, but the main purpose for today lecture is on constraint certification problem on so called CSP. So the overview of today's lecture is I'm going to discuss the problem solving agents, which is a kind of Go based agent. And we are talking about how to formulate a problem so that the AI system can solve the problem and find a solution. And I give you some examples. I talk about basic search algorithms and then I will talking about CSP, which are constraint certification problems. So some example for the problems that the AI system can solve, this is the Apuzzo problem. So the apostle having nine cells. One of them is an empty cells, and then the other eight is labeled with the towel from one to eight. And then they will have some initial configuration in this task state. And the problem is to move the towels, the number tout with only one blank cell here in such a way that after you make a number of moves and then the board will reach this ghost plate. And so the towel will be organized in this way. So if you can find a way to move the towels and then get into this Go state, then you solve the problem. So there is one example of problem solving and with AI algorithms we will be able to solve this kind of problem. Another very relevant problem is so called robot navigation. So just imagine that you are making a robot in order to help people. Let's say that in a nursery to help elderly people, because some of them have mobility difficulty. And so the robot need to help them to move from place to place and then assist them with their daily activities. For the robot to do that, they have to navigate within an environment. Let's say that this is the layout of the area within the nursery with the gray cells essentially denoting the world that the robot cannot step into. And the plant cell is essentially the space where the robot can go into. The robot currently at the start state in this red cell here. And its goal is to reach this Go state in this green cell here, for instance, it need to go to this room, this square, in order to assist the elderly person. So with this one, one way to formulate a problem is to denote the coordinates of different cells within the environment. And so, for instance, the star state will have the coordinates of the S coordinate will be zero and the y coordinate will be three. And so star state is a pair of integer, in this case is three. Then Go state can be represented similarly. And in this particular case, Go state green cells is added coordinate with the S coordinate bin CIS, the y coordinate being two. And so the Go state is a pair of integer CIS two. And so you can see that with this kind of problem, the agent, in this case the robot is trying to solve the problem by finding the path to get the robot from the star state three to the Go state sys two. And again, such algorithm allow you to solve these kind of problems. Clearly this environment is just one of the example environments. So the layout of the area can be much bigger, maybe 1000 by 1000 and there may be hundreds of words and so on and so forth. So those are the two examples on the kind of problems that we hope to use some search algorithm to solve. And for our system to solve those problems with an algorithm, we need to formulate the problems. We will focus on the so called single state problems because they are the easiest kind of problems. Imagine that if the robot's information is not complete, it doesn't really know about the layout of the areas. It can only say that the current state can be one of the three possible locations in the area. Then that become much harder. So that is the multistate problem and we are not dealing with those difficult problems. So we are dealing in this lecture with the simplest kind of problem, the single state problem. And so we will formulate this kind of problem. Now, the problem is defined by four items. So it's consisting of four components. The initial state for instance, in the problem for robot navigation, the initial state is the star coordinate, which is three. Then the agent is equipped with a number of actions it can perform, in particular in that robot navigation problems. Then clearly the robot can go up, go down, go to the left, go to the right. And so those are the possible actions that the robot can perform. Now, even though there are four actions that the robot can perform, but because the robot may be next to a world next to the boundary of the environment at the corner. And so not all actions can be performed. And so the successor function SX allow you to map from a current state x and tell you what are the legitimate actions that the robot can perform. For instance, in the square three, the robot cannot go to the left because it's already at the boundary of the environment, it could only go up, go down and go to the right. And so in this case, the successor functions will say that in the state three, the successor functions define these three pairs. Action up. And then the robot will be able to go from zero three to or the second pair is perform the action down. The robot is able to go from three to four. Or the robot can also choose to go right, performing the action right. And then the robot will go from the state zero three to state 13. So the actions that the robot can perform, the legitimate actions that the robot can perform will be represented by the successor function SX for every state x in the environment. So that's the second component. So the first component, initial state, second component, successor function, the third component is the Go test. So in some problems the Go test can be explicit. For instance, in the robot navigation problems, the Go test is essentially saying that the agent has to be reaching the coordinate sys two, that is the Go state. But in some problems the Go test could be implicit. For instance, if the agent's task is to play chess, then when playing chess, then the agents is winning achieving its goal, it is put its opponent in a checkmate position. So the goal is satisfying that the position of the chessboard x is a checkmate against the opponent. So the Go test define the third component and the last component essentially say that there are many ways to go from the initial state to a Go state and some ways could be better, could be more optimal. And so in order to represent whether one way is better than another, we define the path cost. So getting from the initial state to a Go state require, you require the action to make many steps. So that sequence of steps is called a path, getting the agent from one state to another state. And in order to show that one path is better than another, we review the cost of the path. And so we assume in the single state problem formulation that the path cost is addicted. So that means that if you have each step cost certain cost, then you just sum all the steps and then the cost of all the steps and then you get the cost of the path. And so the assumption is always that the step cost for each step in the past is defy and assumed to be non negative. Because if you can take an action. And then instead of is cost the agents is actually benefit the agents, then that is not considered to be a cost and that would make the problem more difficult to solve. And so we want to have additive and non negative cost. So if you have a problem like this one, then solving the problem essentially means that you produce a solution, which is a sequence of actions leading from the initial state to a goal state. And remember that there may be many possible goal states. For instance, in playing chess, you can have a Go when you still have ten pieces and the opponents may have eleven pieces, but you put the opponent in checkmate. In another situations, when you only have five pieces, the opponents have three pieces, and then you put the opponent in checkmate, you also reach to Go. And so Go state doesn't have to be unique. In the case of the example for robot navigation, you can see that there is only one Go state, CIS two. But in many situations the Go state can be multiple and the job is to find one solution that lead to one Go state. All right? So that is the problem. Now we have been able to formulate it and then put all of the components of the problems, like the initial state, like the successor function, like the Go test and the past cost, into some data structure. And with this data structure, hopefully we are able to have some good algorithms to solve these problems. So the algorithms are put under the so called research algorithms. And so with these research algorithms, there are many of them, not just single one, but they follow the same structure by using the tree data structure in order to execute the search systematically. So let's go back to the robot navigation problem. So the tree search start with the root node of the search tree. And this root node of the search tree will store the initial state. In this case, that is the coordinate three. Now remember that the successor functions from the state three have three legitimate actions. The agent can go up, go down or go right and cannot go left because it's already at the boundary of the environment and so cannot go left. And so with this initial state now expanding this search from this root node of the search, then the system will be able to expand the search and get these three children of this zero three. So if it go up, it can reach the stage two, if it go down, it can reach the stage four, and if it go right, it can reach the stage 13. Now you can see that the tree has an upside down structure. So the root node of the tree is actually sitting on the top, and then we expand it downward. Clearly you can also draw the tree with the root node at the bottom and then expand it upward but then that is just the convention. In this case we choose the root node at the top and expand the tree downward. And the second thing that you will observe is that after a node has been expanded we will shade it with green color while the nodes that haven't been expanded, they will have the black boundary with white inside. And so these nodes are the candidates for expansion. The question is which node should we expand next? So there are three, should we expand zero two or four or one three? And as you will see, choosing the node expand next is actually defining the search strategies. And so let's say that we will choose the left mode nodes to expand and so we choose zero two then the next one is that we will see that from zero two there are two nodes that we can reach. So on the left is the boundary of the environment, on the right is a wall. And therefore, from zero two, you can only go up or go down. Go up, get you to one, go down, get you back to three. And so you can see that and after two has been expanded, then you can see that we again shade it with green color. And then the two new children of the surgery will now be added. One because you can go up from two to one and three because you can go down from zero two back to three. And now again so the list of nodes that are candidates for expansions are these four, this one, this one, this one and this one. Then again the search algorithm faced the questions of these are the four candidates for expansions, which one should it choose to expand next? And that is the essence of the search algorithms. A particular search method will have a way to choose which node to expand next. So now let's look into the data structure that we will use in order to represent the search algorithm and then that will allow us to implement the search algorithm as well. So here you can see that on the one side you have the state of the problem. So this is the state for instance if we are talking about the aperture problem so this is a state, every configuration, every representation of a physical configuration of the problem is a state. From one state you can take an actions and then you move to another state. On the other hand, the search tree consisting of the search node like this one and the search node itself is a data structure. This data structure contain a number of components. Firstly it say that which state in the state space this node is associated with. Secondly, the node data structure will also tell you which is the parent node that from this parent node and then expand the parent node and then you obtain this particular node and then it also say that from the parent node which action has been taken in order to bring you to this current node. So for instance, you go right and so on and so forth. And what is the depth of this node within the search g? So let's say that if you look into this search g with these four nodes, so this is a search g. And let's say that this is the root node of the searchee. Then always the root node have the depth zero. And then clearly this node will have the depth one. This node, we have the depth two. And this node, we have the depth two as well. In this particular case, this is not the root node. And so there will be some root node up here. And the data structure tell us that the depth of this node is sits. And then let's say that every step that you need to take cost one. Then clearly with the depth CIS, then you go from the root node to this node. The path cost will be six as well. Okay? And so this is the example with the Apers of problem. But let's look at the more familiar example. Our initial state is zero three. So this one is clearly the root node. So the depth of this one is zero. Okay? And the cost to reach to this node is also zero. Now from this node you expand and then you get one of the children of this node is two. The action to get from zero three to zero two is go up, okay? The depth is clearly because three is a root node. Then this node here, the green one here, we have the depth one, okay? And then the path cost getting from the initial state zero three to this node taking one step, we have a cost one as well, okay? And then subsequently if you go to this node zero one here, then now you can know that the state associated to this node is the coordinate one. In the environment, the actions going from zero two to zero one is again you go up, okay? Move up to go from zero two to zero one. And then the depth. Now remember that this root node, that's zero, this node depth one and this node now have the depth two, okay? Depth equal to you see that? And clearly the cost, path cost getting from the root node zero three to this zero one is that you take one step here. So cost one another step here. So take cost one again. And so the sum of the cost is again, cost is also equal to two. Okay? So you have cost equal to two here as well. So this is the data structure associated to this node here, okay? And clearly the parent of this particular node, zero one is this node, the green one. Yeah. This is the parents of this zero one. Okay? So with this you can see that a node will contain the state. It is associated with the parent node. From this parent node you expand the parent node and then you get to this node the action that bring you from the parent node to this node. And then the path cost that form the initial state gets in through that path. What would be the cost? And then the depth again from the initial state which is the root node of the search tree, what is the depth of this node? Now let's go back to what I mentioned before. As you expand in the search tree, then there will be a list of node that you consider candidates for expansion. Now that list of node will be stored in the data structure called frontier. Okay? These are the list of generated nodes on the search tree which are not yet expanded because they are not yet expanded. So now they become the candidate for expansion. And so the whole search algorithm so this is the illustration of the frontier, okay? So this is the search tree that we have constructed for the robot navigation problems. So we go from zero three and then we expand it and then we get the three nodes 20413. And then you choose two and then you have three which are the two new children of two. So the frontier contain the generative node which are not yet expanded. That means that this list of nodes 30, four and what? Three here. So this is the frontier simple search tree algorithm is that when you expand the frontier, then you will choose which node to expand. So frontier is a list of node candidates for expansion and a search strategies will allow you to choose which of them to expand. Okay, some popular search strategies. For instance, deferred Search, preface search, digest search best for search such as grid D, bestford Search and a star. These are a number of popular search algorithms that allow you to choose which node from the frontier to expand. There are many more, but these are the more popular ones. And the search algorithm systematically visit the state in the state space until it reaches a state that satisfy the goal test. And clearly after it can reach the goal test, now it know what could be the path to get from the initial state to this state satisfying the Go test. Why? Because every node on the search tree we always store the parent. So if we get to a node that at this node, the state associated with this node, the state s associated with this node happened to satisfy the Go test, then we know that we reached the goal, right? And then from this node we go back to the parents and then we get the parent of this node and then go back to the parents, go back to the parent of this node and then so on and so forth, until at some point we go back to the root node of the search tree, which is the initial state. And therefore this part here or this. Step here, step here, step here, step here, step here, step here and step here form the path gets seen from the initial state to this go state and this sequence of actions is the solution in the process that the algorithm systematically visit the state in the state space. Then the search tree is constructed in the computer memory. You still remember with the robot navigations we start from the initial state s three so there's only one node on the search g and then after we expand this node and then we have three more nodes in the search g, right? So go up and then we get this node go down, we get this node go right away we get this node. So you can see that by doing these expansions the search g keep growing and then they are going to be stored in the memory of the computer when I Go state can be reached and then the sequence of action step from the initial state that I Go state can be reconstructed. This is what I already explained in this process. So solutions file and we return the solution. But if we can never reach a goal state, then we return the statement telling the agent that no solution can be found. So this is essentially the process of using such algorithm to find the solution which is the sequence of action LinkedIn from the initial state to a Go state. A Go state is a state satisfying the Go tag.