**Modified**

All right, so far we look at the problems where the path were the solution. For instance, the robot navigation problem is one of the examples of traveling on a graph, where we from the initial node on the graph and the agents trying to get to some Go node on that same graph. And the problem required the agents to find a path to go between node until it can get to that Go node. And so solving the problem essentially is to find the path to get from the initial node to the Go node.

Or the A puzzle is another example of solving the problem and find the path to get from the initial configurations of the A puzzle to the Go configuration of the A puzzle. And so what are the steps that the agent need to perform in order to get from the initial configuration to the Go configuration.

But in many problems we just want to find a Go state and it doesn't matter how we get there. And so some example of these kind of problems, including the queen's puzzle. So the queen's puzzle is the problem of placing the A queen on a chessboard, so that no two queens attack each other. So the cues are the queens. And so you can see that the queen can attack anything horizontally. So on the same row and vertically so on the same column.

So anything, if you place on this same column and it's same row, then this queen can definitely attack. So this particular queen can attack anything here and here and on this column as well. Not only that, the queen can also attack diagonally. And so anything along this diagonal and this diagonal will be under attack by this queen. Meaning that you cannot place any other queens on this row and this column on all these squares here. If you already put one queen here.

And your job is to put a queens on this chessboard and no two queens attacking each other. Because remember that if one queen can attack another, then clearly the other can also attack that queen as well. So you can see that if you face with this problem, then you don't really need to find any steps. As long as you can find a configuration for the queen on the chessboard, then you achieve the goal.

Another problem is very familiar with some of you. So this sudoku. And so you have this particular board, nine by nine. So nine rows by nine columns. And so they also split into nine subsquare as well. So each of these is one subsquare, each of these one sub square and each of these one sub square.

Now there are 81 squares on this one, on this grid. And now you have number from one to nine. And you put those number into this in such a way that in every row you will have all number from one to nine. And so that means that there's no repeat of any number on any of this row and same for the columns.

So all number from one to nine had to be on the columns and there's no repeat of any number. And the same thing for each of these subsquare. So the square of three by three here, or the square of three by three here, each of them need to have also OD, nine numbers from one to nine. So you can see that each number from one to nine, each number has to appear nine times.

So in each of these subsquare there must be one number, one and only one. And each of these subsquare then need to be exactly one number, two and so on and so forth, and they are nine subsquare. Now, if the problem is well posed, because the problem essentially fills some of the square with some of the numbers and then you need to find out the numbers for the empty squares, not all problems are well posed.

Because let's say that if you put the number in such a way that let's say that if there is no numbers on the whole grid, then there will be potentially many different solutions. So if the problem is well posed, it has only one unit solution and there will be 27 constraints that you need to deal with in such a well posed pseudoku problem.

And again, the steps of how you get to agree with all the numbers is not critical. The important thing is to have all the squares filled by the numbers. As soon as you are able to find the numbers for each and every square on this board, then you solve the problems. You have a solution to the pseudoko problem.

Now, can these problems be solved by search algorithms? Then the short answer is yes. You can always use search algorithm to solve these problems. And so if, let's say that you use search algorithms, then essentially the successor is all valid ways of placing additional queens on the board or all valid ways of putting a number into one of the empty squares.

And then the goal is that all the a queens are going to have its place and then they don't attack each other or in the pseudoku problems, all the empty cells or the empty squares is occupied by one single numbers and they satisfy all the pseudo code constraints.

Now, with these roles and the successor functions, then you should be able to build a search tree as well. So let's just imagine that this is one particular node on the search tree where you already place five queens. And then what you need to do next is to place another five queens.

Then what you could do is to simply find, based on the successor functions, what are the next states from this state. And then you expand the state by adding new nodes to the searchee. And so you are going to be able to add this node to the searches. So you have a new set of unexpended node that you can place into the frontiers.

And then clearly this node now will become expanded and then you need to remove it from the frontier. And if you do this, then the search algorithms will allow you to reach a node on the searches where all the queens have been placed on the board. Then that is when you are able to say that I found a solution to the problem and the sequence of actions are placed in which Queen is not that critical.

Because the goal of this problem is to find the configurations for the queen on the chessboard and not to find the sequence of steps getting from the initial state, which is the empty chessboard, to that go state where all the queens have been placed. The problem is that the tree is going to be very big.

Imagine that if you have the empty board to start with, so there are eight by eight grid and therefore it is 54 squares. Each of these squares you could place the first Queen. And so that means that there are 64 branches under that root node of the surgery. So you can see that the surgery will become huge.

And how deep will the tree be? Clearly you need to place a queens. And so from the initial state, which is the empty chessboard, to the leaf node, where you place all the a queens, there will be eight steps need to be taken. So that means that the death will be eight.

And then the number of branches under each node can be up to 64. So that means that after you have the first initial node, which is at the root of the surgery, then you have 64 children under that node. And then under each of those 64, you can have 63 children. And so you can see that the size of the surgery will become exponential.

And because of that, such formulation of the Queen's puzzle can become very big and it becomes not exactly computationally checkable. And if you use the successors being ovalic ways of placing the Queen in the next column, so you can see that these five queens occupy the first five columns, and then you only need to look into column number six here.

Then clearly by doing this, you can actually reduce the size of the surgery, because instead of having five Durants like on the previous slide, now you are only looking at two children because for the next column, column number six, there are only two places where you could put the new Queen safely on the chessboard.

So you reduce the branching factor of the surgery. Still, the surgery size would still be very big because from the very first initial state, which is the anti chessboard, clearly instead of placing the Queen in any of the empty cells on the search board, on the chessboard which are city four cells. Now you only look at the first column and on the first column there are only eight anticells and therefore the branching factor will become eight and not city four.

Still, the surgery will still become too big to be checked above and allow you to have search algorithm to work efficiently. The next questions if you use search algorithm to solve is that what kind of search is bad? Are you going to use deference search, preferred search a star or what kind of search? And so again it presents you with a number of questions that they are not easy to answer.

And so you can see that with the queen's perspective problems, which is not the most difficult problems out there, then there are already quite a number of decision problems that you need to solve in order to solve the problem efficiently and chapter plate. So instead of using search algorithms for all kind of problems, then people actually looking into the Car problems where the path doesn't matter and all you need is to find the right way to configure the Go state.

Then that will fall under the umbrella of so called constraint certification problem or CSPs. In the standard search problem, the state is a black box, any data structure that supports successor functions, heuristic function and Go test, then you can formulate them under standard search problem.

In the CSP on the other hand, the state is defined by a set of variables. So you have variable s one, s two, s three and so on and so forth. For instance, in the case of the eight queen problems where you place the eight queens on the chessboard, then you have eight variables, variable s one or q one for the queen on the first column, variable q two for the queen on the second column and so on and so forth.

And so with those eight variables you will be able to try to find the values for each of the variables by choosing the values from a domain bi for variable si. For instance, for each of the queen q one, q two to q eight, you can see that they can occupy any of the row from row one to row eight because each queen will be placed on the corresponding column and then each column will have eight possible squares in that column.

And so you will see that the domain the I of each of the queen qi will be a value from one to eight denoting the square in that column between one and eight. Now, the Go test in the CSP is the set of constraints specifying the allowable combinations of values for success of variables. And so these constraints, for instance in the queen, the eight queen problems will specify that if the queens on the same row or on the same column, then they violate the constraints.

So they have to be on different columns, on different rows and they cannot be on the same diagonal either. The CSP is with the state represented by variables taking values from domains. And the Go test specified by a set of constraints is a simple example of a formal representation language.

There are many formal representation languages that people that human use in order to represent different kind of problems, allowing us to look at the solutions by formulating the problem within that representation language. And CSP is one of such representation language.

The important thing with CSP is that it allows useful general purpose algorithms with more power than standard search algorithms. So that means that these algorithms, they are general purpose because they are not specific for any particular specific problem like eight queens or sudoku. They can be used for any CSP, doesn't matter that CSP is an eight queen or a sudoku, but then they are more powerful than the standard search algorithm.

Example of CSP is the map coloring problem. Just imagine that you work for a publishing company and the part of the company is that the Australian government asked the company to print the map of Australia such that each state of Australia will be colored by a different color.

And then because the government tries to save money and so they say that there are only three colors should be used. So Australia prefer to use red, green and blue. And so you don't want to use more than three colors because that would increase the cost. And if the two states next to each other, you want to color them in different colors, otherwise you won't be able to recognize they are two different states.

So adjacent state had to be colored by different colors, and because of this constraint that adjacent state had to be colored by different colors. This can be represented as a CSP, as a constraint satisfaction problem. In this particular CSP, the variables are the state. So was for Western Australia. Is one state, Nt for Northern Territory is another state.

Q for Queensland and SW for Nisaworld. V for Victoria, s A for South Australia and T for Tasmania. So these are different variables representing different states or territories in Australia, the domain di for each and every variable is the allowable colors red, green and blue.

The constraints is that adjacent states must have different colors. And so that means that because Western Australia is adjacent to Northern territory and SA, so there's an arc between WA and SA w A and nt, but because WA is not adjacent to q and therefore there's no arc between WA and q and so with this kind of constraint graph, we are able to represent the binary CSP easily and allow us to use some graph based algorithm to solve the constraint satisfaction problem

**Summarise**

The discussed problems involve scenarios where finding a path is the solution. For example, robot navigation involves finding a path from an initial node to a goal node on a graph. In the case of puzzles like the A puzzle, the focus is on reaching the desired configuration from the initial one. However, some problems only require reaching a specific state, regardless of the path taken. The queen's puzzle involves placing queens on a chessboard so that they don't attack each other. Similarly, Sudoku requires filling a 9x9 grid with numbers while adhering to specific rules.

While search algorithms can solve these problems, they lead to large search trees due to the exponential branching factor. Constraints can help manage the tree size, like placing queens in columns without attacks. This approach reduces branching but remains computationally intensive. Constraint Satisfaction Problems (CSPs) offer a more efficient alternative. In CSPs, variables represent the state, each with a domain of possible values. Constraints define allowable value combinations.

CSPs utilize a formal representation language, enabling general-purpose algorithms with more power than standard searches. An example is the map coloring problem, where adjacent regions need distinct colors. A constraint graph represents binary CSPs, aiding in solving them using graph-based algorithms. CSPs provide a powerful approach to solving a variety of problems efficiently.

***Important***

**SOLVING THE QUEEN'S PUZZLE AND THE PATH**

In many problems we just want to find a Go state and it doesn't matter how we get there. Some example of these kind of problems, including the queen's puzzle. As long as you can find a configuration for the queen on the chessboard, then you achieve the goal.

**SUDOKU AND SEARCH ALGORITHMS**

Can these problems be solved by search algorithms? Then the short answer is yes. You can always use search algorithm to solve these problems. With these roles and the successor functions, you can build a search tree.

**AUSTRALIA IN LAPLIKE 2**

The constraints is that adjacent states must have different colors. One solution is essentially a complete and consistent assignment of colors to variables. This is one valid solution, because each and every region is assigned one color.

**CONSTRAINT GRAPH**

Most of the time, in order to formulate the CSP problem, people use constraint graph. With constraint graph you can represent in particular binary CSP. Each constraint relays only two variables and the arcs are the constraints. This allows us to use some graph based algorithm to solve the constraint satisfaction problem.

**Original**All right, so far we look at the problems where the path were the solution. For instance, the robot navigation problem is one of the examples of traveling on a graph, where we from the initial node on the graph and the agents trying to get to some Go node on that same graph. And the problem required the agents to find a path to go between node until it can get to that Go node. And so solving the problem essentially is to find the path to get from the initial node to the Go node. Or the A puzzle is another example of solving the problem and find the path to get from the initial configurations of the A puzle to the Go configuration of the A puzzle. And so what are the steps that the agent need to perform in order to get from the initial configuration to the Go configuration. But in many problems we just want to find a Go state and it doesn't matter how we get there. And so some example of these kind of problems, including the queen's puzzle. So the queen's puzzle is the problem of placing the A queen on a chessboard, so that no two queens attack each other. So the cues are the queens. And so you can see that the queen can attack anything horizontally. So on the same row and vertically so on the same column. So anything, if you place on this same column and it's same row, then this queen can definitely attack. So this particular queen can attack anything here and here and on this column as well. Not only that, the queen can also attack diagonally. And so anything along this diagonal and this diagonal will be under attack by this queen. Meaning that you cannot place any other queens on this row and this column on all these squares here. If you already put one queen here. And your job is to put a queens on this chessboard and no two queens attacking each other. Because remember that if one queen can attack another, then clearly the other can also attack that queen as well. So you can see that if you face with this problem, then you don't really need to find any steps. As long as you can find a configuration for the queen on the chessboard, then you achieve the goal. Another problem is very familiar with some of you. So this sudoku. And so you have this particular board, nine by nine. So nine rows by nine columns. And so they also split into nine subsquare as well. So each of these is one subsquare, each of these one sub square and each of these one sub square. Now there are 81 squares on this one, on this grid. And now you have number from one to nine. And you put those number into this in such a way that in every row you will have all number from one to nine. And so that means that there's no repeat of any number on any of this row and same for the columns. So all number from one to nine had to be on the columns and there's no repeat of any number. And the same thing for each of these subs square. So the square of three by three here, or the square of three by three here, each of them need to have also OD, nine numbers from one to nine. So you can see that each number from one to nine, each number has to appear nine times. So in each of these subsquare there must be one number, one and only one. And each of these subsquare then need to be exactly one number, two and so on and so forth, and they are nine subsquare. Now, if the problem is well posed, because the problem essentially fills some of the square with some of the numbers and then you need to find out the numbers for the empty squares, not all problems are well posed. Because let's say that if you put the number in such a way that let's say that if there is no numbers on the whole grid, then there will be potentially many different solutions. So if the problem is well posed, it has only one unit solution and there will be 27 constraints that you need to deal with in such a well posed pseudoku problem. And again, the steps of how you get to agree with all the numbers is not critical. The important thing is to have all the squares filled by the numbers. As soon as you are able to find the numbers for each and every square on this board, then you solve the problems. You have a solution to the pseudoko problem. Now, can these problems be solved by search algorithms? Then the short answer is yes. You can always use search algorithm to solve these problems. And so if, let's say that you use search algorithms, then essentially the successor is all valid ways of placing additional queens on the board or all valid ways of putting a number into one of the empty squares. And then the goal is that all the a queens are going to have its place and then they don't attack each other or in the pseudoku problems, all the empty cells or the empty squares is occupied by one single numbers and they satisfy all the pseudo code constraints. Now, with these roles and the successor functions, then you should be able to build a search tree as well. So let's just imagine that this is one particular node on the search tree where you already place five queens. And then what you need to do next is to place another five queens. Then what you could do is to simply find, based on the successor functions, what are the next states from this state. And then you expand the state by adding new nodes to the searchee. And so you are going to be able to add this node to the searches. So you have a new set of unexpended node that you can place into the frontiers. And then clearly this node now will become expanded and then you need to remove it from the frontier. And if you do this, then the search algorithms will allow you to reach a node on the searches where all the queens have been placed on the board. Then that is when you are able to say that I found a solution to the problem and the sequence of actions are placed in which Queen is not that critical. Because the goal of this problem is to find the configurations for the queen on the chessboard and not to find the sequence of steps getting from the initial state, which is the empty chessboard, to that go state where all the queens have been placed. The problem is that the tree is going to be very big. Imagine that if you have the empty board to start with, so there are eight by eight grid and therefore it is 54 squares. Each of these squares you could place the first Queen. And so that means that there are 64 branches under that root node of the surgery. So you can see that the surgery will become huge. And how deep will the tree be? Clearly you need to place a queens. And so from the initial state, which is the empty chessboard, to the leaf node, where you place all the a queens, there will be eight steps need to be taken. So that means that the death will be eight. And then the number of branches under each node can be up to 64. So that means that after you have the first initial node, which is at the root of the surgery, then you have 64 children under that node. And then under each of those 64, you can have 63 children. And so you can see that the size of the surgery will become exponential. And because of that, such formulation of the Queen's puzzle can become very big and it becomes not exactly computationally checkable. And if you use the successors being ovalic ways of placing the Queen in the next column, so you can see that these five queens occupy the first five columns, and then you only need to look into column number six here. Then clearly by doing this, you can actually reduce the size of the surgery, because instead of having five Durants like on the previous slide, now you are only looking at two children because for the next column, column number six, there are only two places where you could put the new Queen safely on the chessboard. So you reduce the branching factor of the surgery. Still, the surgery size would still be very big because from the very first initial state, which is the anti chessboard, clearly instead of placing the Queen in any of the empty cells on the search board, on the chessboard which are city four cells. Now you only look at the first column and on the first column there are only eight anticells and therefore the branching factor will become eight and not city four. Still, the surgery will still become too big to be checked above and allow you to have search algorithm to work efficiently. The next questions if you use search algorithm to solve is that what kind of search is bad? Are you going to use deference search, preferred search a star or what kind of search? And so again it presents you with a number of questions that they are not easy to answer. And so you can see that with the queen's perspective problems, which is not the most difficult problems out there, then there are already quite a number of decision problems that you need to solve in order to solve the problem efficiently and chapter plate. So instead of using search algorithms for all kind of problems, then people actually looking into the Car problems where the path doesn't matter and all you need is to find the right way to configure the Go state. Then that will fall under the umbrella of so called constraint certification problem or CSPs. In the standard search problem, the state is a black box, any data structure that supports successor functions, heuristic function and Go test, then you can formulate them under standard search problem. In the CSP on the other hand, the state is defined by a set of variables. So you have variable s one, s two, s three and so on and so forth. For instance, in the case of the eight queen problems where you place the eight queens on the chessboard, then you have eight variables, variable s one or q one for the queen on the first column, variable q two for the queen on the second column and so on and so forth. And so with those eight variables you will be able to try to find the values for each of the variables by choosing the values from a domain bi for variable si. For instance, for each of the queen q one, q two to q eight, you can see that they can occupy any of the row from row one to row eight because each queen will be placed on the corresponding column and then each column will have eight possible squares in that column. And so you will see that the domain the I of each of the queen qi will be a value from one to eight denoting the square in that column between one and eight. Now, the Go test in the CSP is the set of constraints specifying the allowable combinations of values for success of variables. And so these constraints, for instance in the queen, the eight queen problems will specify that if the queens on the same row or on the same column, then they violate the constraints. So they have to be on different columns, on different rows and they cannot be on the same diagonal either. The CSP is with the state represented by variables taking values from domains. And the Go test specified by a set of constraints is a simple example of a formal representation language. There are many formal representation languages that people that human use in order to represent different kind of problems, allowing us to look at the solutions by formulating the problem within that representation language. And CSP is one of such representation language. The important thing with CSP is that it allows useful general purpose algorithms with more power than standard search algorithms. So that means that these algorithms, they are general purpose because they are not specific for any particular specific problem like eight queens or sudoku. They can be used for any CSP, doesn't matter that CSP is an eight queen or a sudoku, but then they are more powerful than the standard search algorithm. Example of CSP is the map coloring problem. Just imagine that you work for a publishing company and the part of the company is that the Australian government asked the company to print the map of Australia such that each state of Australia will be colored by a different color. And then because the government tries to save money and so they say that there are only three colors should be used. So Australia prefer to use red, green and blue. And so you don't want to use more than three colors because that would increase the cost. And if the two states next to each other, you want to color them in different colors, otherwise you won't be able to recognize they are two different states. So adjacent state had to be colored by different colors, and because of this constraint that adjacent state had to be colored by different colors. This can be represented as a CSP, as a constraint satisfaction problem. In this particular CSP, the variables are the state. So was for Western Australia. Is one state, Nt for Northern Territory is another state. Q for Queensland and SW for Nisaworld. V for Victoria, s A for South Australia and T for Tasmania. So these are different variables representing different states or territories in Australia, the domain di for each and every variable is the allowable colors red, green and blue. The constraints is that adjacent states must have different colors. And so that means that because Western Australia is next to its adjacent to Northern territory and so WA has to be different color from Nt or WA anti in red, green, red, blue, green, red, green, blue, blue, red, blue, green and so any of these configuration is allowed. But if you put red red for W-A-N-T then you violate the constraint and same for every order pair of adjacent states like WA and SA or Nt and SA, q for NNT, Q and SA, Q and SW and so on and so forth. So one solution is essentially a complete and consistent assignment of colors to variables. And so if you have managed to assign some colors to each of the states and then it doesn't violate the constraint that two adjacent states need to have different colors, then you find a solution. One possible solution is that Western Australia is red, Nt is green, Q is red, NSW is green, V is red, SA is blue, and T is green. So this is one valid solution, because each and every region is assigned one color and the colors are consistent, they don't violate the constraints because two adjacent regions don't share the same color. Now, most of the time, in order to formulate the CSP problem, people use constraint graph. So with constraint graph you can represent in particular binary CSP. So each constraint relays only two variables and the constraint graph essentially consisting of the nodes being the variables. So the nodes on this graph, you can see that they are variables like QSA and so on and so forth. And the arcs are the constraints. For instance, because WA is adjacent to nt and SA, so there's an arc between WA and SA w A and nt, but because WA is not adjacent to q and therefore there's no arc between WA and q and so with this kind of constraint graph, we are able to represent the binary CSP easily and allow us to use some graph based algorithm to solve the constraint satisfaction problem.