**Modified**

So, from the previous slide I mentioned about, there is another technique to provide early detection for the failures. So after we assign, we assign red to WA and then we assign green to green land queue, you can see that the data structure that we use to store all the remaining legal values for the remaining variables tell us that the remaining legal value for Nt is blue and the remaining legal value for SA is blue. And because they are adjacent to each other, so they cannot be of both same values. This particular technique to detect early detection, to provide early detection for the failures is called constraint propagation, which is one powerful technique for CSP. And generally, if you propagate a constraint repeatedly, you can actually enforce the constraints locally because you propagate the constraints from one variable to its related variables and keep doing that.

So this technique in general can be implemented in many different constraint propagation methods. The simplest form of constraint propagation called ask consistency. So ask consistency is essentially by looking to two related variable x and y. This propagation from x to y is considered to be consistent if and only if for every value x of x there is some allowed value for y.

Let's say that this is the current search process to do the map coloring for Australia. And then we start with WAB in red, and then we call Queensland with green. Now, this is the data structure storing all the remaining legal values for the remaining unassigned variables. So, if we look into the variable SA, we can see that the only values the SA can check is blue. And our consistency allow us to say that because SA is adjacent to new sour wells and therefore these constraints, SA being blue, will be propagated to New South Wales. And so now new sour wells can no longer be blue and so it will have to be only red. Now, after we eliminate the blue color from mutual wealth, then this can continue to be propagated to chasen variable for mutual wealth. So when a variable x lose a value, then the neighbor of x need to also be rechecked. So after Nissan wells lost the value blue, now we know that NASA wells can only have the value red and therefore it propagates this constraint to Victoria, which is next to NASA wells. And therefore Victoria can no longer take the value red.

And so by doing this consistency propagation, then we can now eliminate these two value from Nuiso well and Victoria. Not only eliminate this value from NASA well in Victoria you can see that by doing these consistency propagations now we can already identify that because SA can only be blue. So it threw out any legal value for Nt. So that means that there is no acceptable value for Nt at all. And by doing this, we know that this current assignment of WA being red and Q being green is not part of a consistent solution and therefore we actually can detect the failure much earlier than forward checking. So we don't wait until we assign a variable with some values and then seeing that there is some other variables with no legitimate values and then we start back checking. But here we could already back check from these points and then decide that we will need to choose another color for Queensland. And so we have to go back to this node on the search tree.

Now I'm going to cover the topic of local search for CSPs. So local search with CSP can be implemented using a number of strategies such as hill climbing, simulated annulin, and they typically work with complete state. That means that all variables aside, so to apply to CSP, what we do is that we actually allow some state with unsatisfied constraints, meaning that we assign value to variables and don't require that all the constraints will be satisfied. And the local search will try to reassign some variable values to reduce the number of unsatisfied constraints. And by doing that, and then we continue to repeat this process until at some point hopefully, we actually eliminate all unsatisfied constraints and then we find a solution.

So the idea here is when we try to reassign the values for some variables to reduce the number of unsatisfied constraints, we typically randomly select any conflicting variables. Or if we have some good heuristic in order to choose it, then one of them is so called the mean conflict heuristic. The mean conflict heuristic actually choose some of the values that violates the fewest constraints amongst the possible values that you can choose for a conflicted, variables in Hue climb mean you use the heuristic function HN that captures the total number of violated constraints. And then with this HN function you will be able to calculate whenever you choose a variable or a values whether they actually have the minimum conflicted constraint.

So for an example of local search, let's look at the four queen problems. Now, with four queen in four columns we have the number of states within four to the power four, which is 256 states. So it is relatively large problems. Now the action is to move queen in a particular column and then the Go test is there is no attack between the queens. The evaluation is the number of attacks. So if we start by randomly put the queens on the chessboard, for instance, we do the first random assignment of values to the four queens. Now we have them positioned like this and then the number of attack being one here, another here, another here, another here, and another here. So altogether we have five attacks. If we choose this queen to reassign the values, then taking any other values like this one, like this one, we were not having the minimum number of conflicts. And so we put the queen on these locations and then the number of conflicts now reduced to two because now we only have this conflict and this conflict. And so in order to resolve these two conflicts and then we choose to move this queen, because move any other two queens, then only reduce, then you will still have at least one conflict. If we remove this queen, then we can resolve both of these conflicts. And as it happens, if we put that queen here, then we actually have a zero conflicts and then we actually get a solution to the problem. So given a random initial state, we can solve the end queens in almost constant time for arbitrary and with high probabilities. For instance, if we have n equal to 10 million, we could still solve the problem quite quickly, so it doesn't really go into the exponential state.

In summary, CSP are a special kind of problems because the states can be decomposed into the numbers of the values of fixed sets of variables. And so you have the state defined by the set of variables and then each state is defined by assigning some value to those variables and the Go test defined by the constraints on the variable values. The simplest algorithm for solving CSPs is by using back checking. So it is that first search with one variable assigned per node and then if you cannot really find the assignments of a value to a variable at that node, then you back check to the previous layers of the search tree.

Now there are a number of heuristics to help you solve this problem more efficiently. So variable ordering and value selection heuristics can help significantly. And if you also store an additional data structure to perform forward check in, then you can prevent assignments that guarantee later failure as well. More powerful than forward checking is so called constraint propagations. For instance consistency to allow you to actually propagate the constraints from one variables to the related variables and in the process actually can provide detection of early detection of failure. And in general, mean conflict is a very useful and powerful heuristic that can be used very effectively in practice to solve many large and large constraints defection problems. Thank you for your attention.

**Summarise**

In the previous section, we explored the concept of constraint satisfaction problems (CSPs) and the use of backtracking with forward checking to solve them. However, there are more efficient techniques to enhance CSP solving. One such approach is constraint propagation, which involves applying constraints repeatedly to enforce them locally. This helps identify failures early in the process and optimize the search.

Constraint propagation is implemented through methods like arc consistency, which ensures that for every value of a variable, there's an allowed value for related variables. This process involves repeatedly propagating constraints from one variable to connected variables, leading to earlier failure detection.

Local search methods for CSPs, such as hill climbing and simulated annealing, aim to reduce unsatisfied constraints. These methods operate with incomplete states, allowing for unsatisfied constraints, and iteratively adjust variable values to improve the situation.

The mean conflict heuristic is a powerful tool in local search. It selects values that violate the fewest constraints, aiding in decision-making during the search process. For example, solving the N-Queens problem using local search involves strategically placing queens on the board to minimize conflicts.

The efficiency of these methods depends on their ability to quickly reach solutions. Local search can work rapidly even with large problems, making it a practical approach. In summary, constraint propagation and local search techniques greatly enhance CSP solving by efficiently identifying failures, iteratively improving solutions, and employing effective heuristics.

Top of Form

***Important***

**CONFIRMATORY CONSTRAINT PROPAGATION**

There is another technique to provide early detection for the failures. This is called constraint propagation, which is one powerful technique for CSP. If you propagate a constraint repeatedly, you can actually enforce the constraints locally.

**LOCAL SEARCH FOR CSPS**

Local search with CSP can be implemented using a number of strategies such as hill climbing, simulated annulin, and they typically work with complete state. The simplest algorithm for solving CSPs is by using back checking. variable ordering and value selection heuristics can help significantly.

**Original**

So, from the previous slide I mentioned about, there is another technique to provide early detection for the failures. So after we assign, we assign red to WA and then we assign green to green land queue, you can see that the data structure that we use to store all the remaining legal values for the remaining variables tell us that the remaining legal value for Nt is blue and the remaining legal value for SA is blue. And because they are adjacent to each other, so they cannot be of both same values. This particular technique to detect early detection, to provide early detection for the failures is called constraint propagation, which is one powerful technique for CSP. And generally, if you propagate a constraint repeatedly, you can actually enforce the constraints locally because you propagate the constraints from one variable to its related variables and keep doing that. So this technique in general can be implemented in many different constraint propagation methods. The simplest form of constraint propagation called ask consistency. So ask consistency is essentially by looking to two related variable x and y. This propagation from x to y is considered to be consistent if and only if for every value x of x there is some allowed value for y. So let's say that this is the current search process to do the map coloring for Australia. And then we start with WAB in red, and then we call Queensland with green. Now, this is the data structure storing all the remaining legal values for the remaining unassigned variables. So, if we look into the variable SA, we can see that the only values the SA can check is blue. And our consistency allow us to say that because SA is adjacent to new sour wells and therefore these constraints, SA being blue, will be propagated to New South Wales. And so now new sour wells can no longer be blue and so it will have to be only red. Now, after we eliminate the blue color from mutual wealth, then this can continue to be propagated to chasen variable for mutual wealth. So when a variable x lose a value, then the neighbor of x need to also be rechecked. So after Nissan wells lost the value blue, now we know that NASA wells can only have the value red and therefore it propagates this constraint to Victoria, which is next to NASA wells. And therefore Victoria can no longer take the value red. And so by doing this consistency propagation, then we can now eliminate these two value from Nuiso well and Victoria. Not only eliminate this value from NASA well in Victoria you can see that by doing these consistency propagations now we can already identify that because SA can only be blue. So it threw out any legal value for Nt. So that means that there is no acceptable value for Nt at all. And by doing this, we know that this current assignment of WA being red and Q being green is not part of a consistent solution and therefore we actually can detect the failure much earlier than forward checking. So we don't wait until we assign a variable with some values and then seeing that there is some other variables with no legitimate values and then we start back checking. But here we could already back check from these points and then decide that we will need to choose another color for Queensland. And so we have to go back to this node on the search tree. Now I'm going to cover the topic of local search for CSPs. So local search with CSP can be implemented using a number of strategies such as hill climbing, simulated annulin, and they typically work with complete state. That means that all variables aside, so to apply to CSP, what we do is that we actually allow some state with unsatisfied constraints, meaning that we assign value to variables and don't require that all the constraints will be satisfied. And the local search will try to reassign some variable values to reduce the number of unsatisfied constraints. And by doing that, and then we continue to repeat this process until at some point hopefully, we actually eliminate all unsatisfied constraints and then we find a solution. So the idea here is when we try to reassign the values for some variables to reduce the number of unsatisfied constraints, we typically randomly select any conflicting variables. Or if we have some good heuristic in order to choose it, then one of them is so called the mean conflict heuristic. The mean conflict heuristic actually choose some of the values that violates the fewest constraints amongst the possible values that you can choose for a conflicted, variables in Hue climb mean you use the heuristic function HN that captures the total number of violated constraints. And then with this HN function you will be able to calculate whenever you choose a variable or a values whether they actually have the minimum conflicted constraint. So for an example of local search, let's look at the four queen problems. Now, with four queen in four columns we have the number of states within four to the power four, which is 256 states. So it is relatively large problems. Now the action is to move queen in a particular column and then the Go test is there is no attack between the queens. The evaluation is the number of attacks. So if we start by randomly put the queens on the chessboard, for instance, we do the first random assignment of values to the four queens. Now we have them positioned like this and then the number of attack being one here, another here, another here, another here, and another here. So altogether we have five attacks. If we choose this queen to reassign the values, then taking any other values like this one, like this one, we were not having the minimum number of conflicts. And so we put the queen on these locations and then the number of conflicts now reduced to two because now we only have this conflict and this conflict. And so in order to resolve these two conflicts and then we choose to move this queen, because move any other two queens, then only reduce, then you will still have at least one conflict. If we remove this queen, then we can resolve both of these conflicts. And as it happens, if we put that queen here, then we actually have a zero conflicts and then we actually get a solution to the problem. So given a random initial state, we can solve the end queens in almost constant time for arbitrary and with high probabilities. For instance, if we have n equal to 10 million, we could still solve the problem quite quickly, so it doesn't really go into the exponential state. In summary, CSP are a special kind of problems because the states can be decomposed into the numbers of the values of fixed sets of variables. And so you have the state defined by the set of variables and then each state is defined by assigning some value to those variables and the Go test defined by the constraints on the variable values. So if the variable values satisfy all the constraints and then you satisfy the Go test. The simplest algorithm for solving CSPs is by using back checking. So it is that first search with one variable assigned per node and then if you cannot really find the assignments of a value to a variable at that node, then you back check to the previous layers of the search tree. Now there are a number of heuristics to help you solve this problem more efficiently. So variable ordering and value selection heuristics can help significantly. And if you also store an additional data structure to perform forward check in, then you can prevent assignments that guarantee later failure as well. More powerful than forward checking is so called constraint propagations. For instance consistency to allow you to actually propagate the constraints from one variables to the related variables and in the process actually can provide detection of early detection of failure. And in general, mean conflict is a very useful and powerful heuristic that can be used very effectively in practice to solve many large and large constraints defection problems. Thank you for your attention.