**Modified**

So your next question would be whether there are different kinds of constraints, differential problems and yes there are. The general way to classify CSPs is by looking at the kind of variables the CSP contains. So the easier kind of CSPs involve only discrete variables and the variables domains are finite. So if you have N variables and then each variable has only finite number of values, then the domain size denoted by D, then we are looking at the number of complete assignments would be in the order of D to the power of N. For example, if we are looking at the simplest kind of CSP in which all variables are boolean So having only exactly two values, true or false? So the domain size is two. And if we have N variables, boolean variables like that, then we are looking at the boolean CSPs and they would include the boolean satisfiability problems with boolean CSP. Even though they are the simplest kind of CSP they are injectable because the complexity in the worst case for solving those kind of CSP is NP complete, which is injectable.

Now this is the simplest kind of CSP where the number of variables n having only finite domains. So each variables have only a finite number of values. It can take now still within the discrete variables. But if the variables have infinite domains, for instance the domain of the variable could be integers or could be strings. Example is for instance in the job scheduling problems where the variables are the start days for each job and end days for each job. Then the start date and end days can be represented as integers and integers. Clearly they have infinite number of values so they can be any numbers. And if you have such discrete variable CSP whose variables have infinite domains like integers, then you will need a constraint language to express something like the study of the chop one plus five has to be less than or equal to study of the chop three. And so starch of one and starch of threes are the variables and they are constrained by this particular constraint where starch of one plus five less than or equals touch of three. Or if the domain of the variable being strings, then just imagine that you are solving the problem of finding the passwords and the password can be any string of characters. Then you can see that the strings can have any numbers of length, any length and also they can be any particular characters, could be from alphabet or could be between zero and nine or could be any special characters on your keyboard. Then clearly the domain is very large, essentially infinite. And so to solve the problems you are dealing with very large domain and the problem would be a lot more difficult than CSP with discrete variables from finite domains and even the easiest kind of CSP for boolean CSPs then the problem is already in chargeable with the complexity of NP complete. So you could imagine that the complexity of more difficult CSPs, such as CSP with discrete variables whose domains are infinite, then would be a lot more complex, but they are not the most complex kind of CSP, because you can also have to deal with CSP whose variable are continuous.

Example is that if you need to schedule the Hubble Space Telescope observations, because this is a very expensive piece of equipment, and then many astrophysics around the world try to access to this telescope and so they have to compete on star and n times. And to schedule these observations, then you have to deal with this time n times, which is continuous values. And so for continuous variables, a lot of the time people will try to formulate them as linear constraints, solvable in polynomial time by linear programming. And so even though they are continuous variables, but then if you can formulate them as linear constraints, then they can actually be solved in polynomial time.

Now, so that is the varieties of the variables in the CSPs. But another way to classify the CSPs is to look at the varieties of the constraints. So there are different kind of constraints with unary constraint being simplest, because unary constraint involve only a single variable. For instance, if you have the variable SA, and SA cannot be green, because for whatever the reason you cannot call the SA by green, then you have a unary constraint. When SA is not equal to green is a constraint that need to be satisfied. The binary constraints, we involve two variables. And so in binary constraints, you have exactly two variables, for instance, you have SA and WA. And because they are adjacent, so you don't want them to be of the same color. And so you use the binary constraints, SA is not equal to WA in order to insert this guy or binary constraint.

Now you also have higher order constraints, involve more than two variables, so it could be three, four, five, and so some of these higher order constraints, including crypt arithmetic, column constraints, and we are going to look at some of these example of crypt arithmetic column constraints. So, example of crypt arithmetic is that let's say that the problem to solve is to find the digits value for each of the letters in this cryptorithic problem. So two plus two, and then you get four. And so each of the letter TW need to be assigned a digit from zero to nine. Okay? And clearly, for you to solve these problems, you may also need to deal with the problems. That two digits. When they add together, they can get a value that is bigger than ten, bigger than nine, and therefore it will result in one digit for the modular of that sum and then another digit for the values that that sum carry to the next column. And so for instance, if O being CIS, then CIS plus C equal twelve, so R will be equal two, but then you will have a value that carry over to the column for W plus W. Okay? So because of that we will use s one, s two and s three to denote the value that carry from one column to the next one. So for instance, s one is for the value that carry from the column O to the column W, okay? So that means that if you have O plus O, then you have is equal to R plus ten times s one, okay? And similarly with the column WW together and then result in U, then you will have W plus W plus the value that carry over from the previous column. So that s one will be equal to U plus ten times s two and similarly to T. So t plus t plus the value the carryover form W plus W to the s two, and then you get the value O plus ten times s three because of these carryover values s one, S two, S three.

Now the variables in this problem will be F-T-U-W ro, s one, S two, S three. And the domain is that each of these letter can take one of these reaches from zero to nine. So that is the domain for each of those variables and then these are the constraints. So these constraints is straightforward. The only particular constraint is that in this crypt arithmetic, each of the letter must receive one unique values. So you cannot have the same digits for O and W for instance. And so because of that, we will use the constraint all div. So that means that all of these variables need to be different from each other. So you cannot have s have the same value as any of the other variables and the same for T, same for U, same for W, same for R, and same for O. So all these saying that all these variables need to take different values. And after you have this particular formulations, now you can formulate them using this constraint graph. And in the constraint graph you can see that some of these constraints are not binary because for instance, this audits involve 123456 variables or these constraints s one plus W plus W equal U plus ten, s two involve 1234 variables. And because of that we will use this kind of notation. So we don't just connect the variables together in the binary constraints, but we use this particular node, these squares, in order to allow you to represent the constraints. Each square is one constraint and then these constraints allow you to connect with the corresponding variable within the constraints. So the top constraints here is the audif constraints. On the other hand, these four box, these four squares here are the other four constraints. And with this we are able to form a constraints graph in order to represent this crypto arithmetic problem.

CSPs are not just for toy problems like N queen's problems or the pseudoko problems on the crypto arithmetic problems. They are actually very useful and they have been used in the industry by many different business and companies. For instance, the staff assignment problem, who teaches what unit, which software developer does what tasks in a software project? These are the staff assignment problems that can be formulated as a CSP. The time tampling problem, for instance, which class is offered, when and where in which room and at what time. So the timetampling problems can also be formulated as a CSP. The transportation scheduling, for instance, which truck go into what destinations, what address to deliver which parcels. So this transportation scheduling has been formulated as ESP factory scheduling, for instance, scheduling of the jobs, the start time, the end time, the jobs, and so on and so forth. You can already notice that you might already notice that many real problems involve real value variables. For instance, the transportation scheduling, the schedule is based on time and time is a continuous real value variable. So how to solve them? The CSP solving CSP to start with let's start with using search algorithms and formulate the CSP solving and a search problem formulation.

So let's start with the straightforward approach and then we can fix it, right? So the straightforward approach is that if we can just formulate the CSP as a search problem formulation, then we will be able to use our search algorithms like we mentioned before, such as Defer search, prefer Search Digest in order to solve CSP and we don't have to worry too much about CSP anymore. For instance, we can formulate the initial state being the empty assignment because we start with no variables being assigned any values. So the empty assignment will be the root node of the search tree, the successor function assign a value to an unassigned variable. And so if a variable has not been assigned a value and then we will try to find some value for that unassigned variable. And then clearly if there's no legal assignment for that unassigned variable, then that particular branch of the search tree is failed. And then so we probably will need to come back and then find another branch of the search tree where the successor function can apply. The Go test essentially is where we reach a date when the current assignment is complete. So that means that all variables have been assigned a value. And because along the process we always make sure that the next very unassigned variable that is assigned a value does not conflict with the current assignment. So that means that it implies that when we can find the complete assignment, it's also consistent as well. It does not violate any of the constraints. And because this process is the same for all CSP, so we should be able to solve problems for discrete value variables.

CSP clearly the path is irrelevant because we don't care which variable that we assign at which step. So we can start with assigning variable s one or we can start by assigning variable s ten. These steps are not relevant to us. All we care is to reach an assignment that is complete and consistent so we can also use a complete state formulation. So now once we have formulated the CSP solving problem as the search problem then back check in search can be used in order to solve the CSP problem. So in Beijing search, this is the deferred search process using single variable assignment. And the idea is that you go defer a branch of the search tree at each step. You look at one variable that has not been assigned, and then you try to find the values for that variable that does not violate any of the constraints. And then you continue to go down the search tree until if at some point you are able to find a value for the unassigned variables, then you know that that process cannot be continued under that branch. Okay? So there are many nodes that you haven't been explored in the process. And so you go down defer search that one branch but then at some point when you reach a node, when you cannot really find any legitimate assignments for this next variable anymore, then you have to back check to the note above it and then try to find a node assignment of that variables and so on and so forth. And at some point maybe you will need to back check to this one and then have to find the solution under this branch for instance. And so this process is the back checking search to solve using deficit search to solve CSP. Back checking search is the basic uninformed algorithm for CSB and it can solve the n queen problems for n at approximately 25. So if you have the ball having 24 rows and 25 columns then this end queen problem can be solved using simple vegeting search and then above 25 then the search tree becomes too big to be handled by a vegetable search.

Now, because this particular technique is uninformed, the question is can our search be better informed? So before we answer the questions, let's look at the bejekin example for the Mac coloring problem for Australia. So we start with the empty assignment so none of the variable has been assigned any of the colors then we choose one state, one region to color it. Let's say that we start with Western Australia. Remember we have three colors red, green and blue. And so in the first branch we call WA by red, in the second branch green and the third graph blue. And then we never search by continue to explore this first branch here and then try to color the next one color the next one Nt. And then because nt is adjacent to WA and therefore nt can only take one of the two color green or blue. And so under this branch now, we have been green for this one, and then the next variable to try is Queensland Q. And let's say that in this particular case, because Queensland is adjacent to Nt and so Nt is already green, so Queensland can only be red or blue.

Top of Form

**Summarise**

The concept of Constraint Satisfaction Problems (CSPs) involves classifying them based on variables and constraints. CSPs can have different kinds of variables, such as discrete with finite domains, discrete with infinite domains (like integers or strings), and continuous. Constraints can be unary, binary, or higher order. For example, solving a cryptarithmetic puzzle involves assigning digits to letters while satisfying arithmetic constraints.

CSPs have real-world applications like staff assignments, timetabling, and scheduling. They can involve variables with real values, like time. To solve CSPs, search algorithms can be used. Backtracking search is a basic approach, but its efficiency is limited. Informed search strategies aim to improve the efficiency by making informed decisions.

For instance, in solving the map coloring problem, backtracking involves trying different colors for regions, but informed search can make better decisions based on adjacent regions' colors. While CSPs pose challenges, they're essential for problem-solving in various industries.

Top of Form

***Important***

**CLASSIFYING CSPS**

The easier kind of CSPs involve only discrete variables and the variables domains are finite. But if the variables have infinite domains, for instance the domain of the variable could be integers or could be strings. There are different kinds of constraints with unary constraint being simplest.

**BACK CHECKING SEARCH TO SOLVE CSP**

In Beijing search, this is the deferred search process using single variable assignment. Back checking search is the basic uninformed algorithm for CSB. Can our search be better informed?

**Original**

So your next question would be whether there are different kinds of constraints, differential problems and yes there are. The general way to classify CSPs is by looking at the kind of variables the CSP contains. So the easier kind of CSPs involve only discrete variables and the variables domains are finite. So if you have N variables and then each variable has only finite number of values, then the domain size denoted by D, then we are looking at the number of complete assignments would be in the order of D to the power of N. For example, if we are looking at the simplest kind of CSP in which all variables are boolean So having only exactly two values, true or false? So the domain size is two. And if we have N variables, boolean variables like that, then we are looking at the boolean CSPs and they would include the boolean satisfiability problems with boolean CSP. Even though they are the simplest kind of CSP they are injectable because the complexity in the worst case for solving those kind of CSP is NP complete, which is injectable. Now this is the simplest kind of CSP where the number of variables n having only finite domains. So each variables have only a finite number of values. It can take now still within the discrete variables. But if the variables have infinite domains, for instance the domain of the variable could be integers or could be strings. Example is for instance in the job scheduling problems where the variables are the start days for each job and end days for each job. Then the start date and end days can be represented as integers and integers. Clearly they have infinite number of values so they can be any numbers. And if you have such discrete variable CSP whose variables have infinite domains like integers, then you will need a constraint language to express something like the study of the chop one plus five has to be less than or equal to study of the chop three. And so starch of one and starch of threes are the variables and they are constrained by this particular constraint where starch of one plus five less than or equals touch of three. Or if the domain of the variable being strings, then just imagine that you are solving the problem of finding the passwords and the password can be any string of characters. Then you can see that the strings can have any numbers of length, any length and also they can be any particular characters, could be from alphabet or could be between zero and nine or could be any special characters on your keyboard. Then clearly the domain is very large, essentially infinite. And so to solve the problems you are dealing with very large domain and the problem would be a lot more difficult than CSP with discrete variables from finite domains and even the easiest kind of CSP for boolean CSPs then the problem is already in chargeable with the complexity of NP complete. So you could imagine that the complexity of more difficult CSPs, such as CSP with discrete variables whose domains are infinite, then would be a lot more complex, but they are not the most complex kind of CSP, because you can also have to deal with CSP whose variable are continuous. Example is that if you need to schedule the Hubble Space Telescope observations, because this is a very expensive piece of equipment, and then many astrophysics around the world try to access to this telescope and so they have to compete on star and n times. And to schedule these observations, then you have to deal with this time n times, which is continuous values. And so for continuous variables, a lot of the time people will try to formulate them as linear constraints, solvable in polynomial time by linear programming. And so even though they are continuous variables, but then if you can formulate them as linear constraints, then they can actually be solved in polynomial time. Now, so that is the varieties of the variables in the CSPs. But another way to classify the CSPs is to look at the varieties of the constraints. So there are different kind of constraints with unary constraint being simplest, because unary constraint involve only a single variable. For instance, if you have the variable SA, and SA cannot be green, because for whatever the reason you cannot call the SA by green, then you have a unary constraint. When SA is not equal to green is a constraint that need to be satisfied. The binary constraints, we involve two variables. And so in binary constraints, you have exactly two variables, for instance, you have SA and WA. And because they are adjacent, so you don't want them to be of the same color. And so you use the binary constraints, SA is not equal to WA in order to insert this guy or binary constraint. Now you also have higher order constraints, involve more than two variables, so it could be three, four, five, and so some of these higher order constraints, including crypt arithmetic, column constraints, and we are going to look at some of these example of crypt arithmetic column constraints. So, example of crypt arithmetic is that let's say that the problem to solve is to find the digits value for each of the letters in this cryptorithic problem. So two plus two, and then you get four. And so each of the letter TW need to be assigned a digit from zero to nine. Okay? And clearly, for you to solve these problems, you may also need to deal with the problems. That two digits. When they add together, they can get a value that is bigger than ten, bigger than nine, and therefore it will result in one digit for the modular of that sum and then another digit for the values that that sum carry to the next column. And so for instance, if O being CIS, then CIS plus C equal twelve, so R will be equal two, but then you will have a value that carry over to the column for W plus W. Okay? So because of that we will use s one, s two and s three to denote the value that carry from one column to the next one. So for instance, s one is for the value that carry from the column O to the column W, okay? So that means that if you have O plus O, then you have is equal to R plus ten times s one, okay? And similarly with the column WW together and then result in U, then you will have W plus W plus the value that carry over from the previous column. So that s one will be equal to U plus ten times s two and similarly to T. So t plus t plus the value the carryover form W plus W to the s two, and then you get the value O plus ten times s three because of these carryover values s one, S two, S three. Now the variables in this problem will be F-T-U-W ro, s one, S two, S three. And the domain is that each of these letter can take one of these reaches from zero to nine. So that is the domain for each of those variables and then these are the constraints. So these constraints is straightforward. The only particular constraint is that in this crypt arithmetic, each of the letter must receive one unique values. So you cannot have the same digits for O and W for instance. And so because of that, we will use the constraint all div. So that means that all of these variables need to be different from each other. So you cannot have s have the same value as any of the other variables and the same for T, same for U, same for W, same for R, and same for O. So all these saying that all these variables need to take different values. And after you have this particular formulations, now you can formulate them using this constraint graph. And in the constraint graph you can see that some of these constraints are not binary because for instance, this audits involve 123456 variables or these constraints s one plus W plus W equal U plus ten, s two involve 1234 variables. And because of that we will use this kind of notation. So we don't just connect the variables together in the binary constraints, but we use this particular node, these squares, in order to allow you to represent the constraints. Each square is one constraint and then these constraints allow you to connect with the corresponding variable within the constraints. So the top constraints here is the audif constraints. On the other hand, these four box, these four squares here are the other four constraints. And with this we are able to form a constraints graph in order to represent this crypto arithmetic problem. So CSPs are not just for toy problems like N queen's problems or the pseudoko problems on the crypto arithmetic problems. They are actually very useful and they have been used in the industry by many different business and companies. For instance, the staff assignment problem, who teaches what unit, which software developer does what tasks in a software project? These are the staff assignment problems that can be formulated as a CSP. The time tampling problem, for instance, which class is offered, when and where in which room and at what time. So the timetampling problems can also be formulated as a CSP. The transportation scheduling, for instance, which truck go into what destinations, what address to deliver which parcels. So this transportation scheduling has been formulated as ESP factory scheduling, for instance, scheduling of the jobs, the start time, the end time, the jobs, and so on and so forth. You can already notice that you might already notice that many real problems involve real value variables. For instance, the transportation scheduling, the schedule is based on time and time is a continuous real value variable. So how to solve them? The CSP solving CSP to start with let's start with using search algorithms and formulate the CSP solving and a search problem formulation. So let's start with the straightforward approach and then we can fix it, right? So the straightforward approach is that if we can just formulate the CSP as a search problem formulation, then we will be able to use our search algorithms like we mentioned before, such as Defer search, prefer Search Digest in order to solve CSP and we don't have to worry too much about CSP anymore. So for instance, we can formulate the initial state being the empty assignment because we start with no variables being assigned any values. So the empty assignment will be the root node of the search tree, the successor function assign a value to an unassigned variable. And so if a variable has not been assigned a value and then we will try to find some value for that unassigned variable. And then clearly if there's no legal assignment for that unassigned variable, then that particular branch of the search tree is failed. And then so we probably will need to come back and then find another branch of the search tree where the successor function can apply. The Go test essentially is where we reach a date when the current assignment is complete. So that means that all variables have been assigned a value. And because along the process we always make sure that the next very unassigned variable that is assigned a value does not conflict with the current assignment. So that means that it implies that when we can find the complete assignment, it's also consistent as well. It does not violate any of the constraints. And because this process is the same for all CSP, so we should be able to solve problems for discrete value variables. CSP clearly the path is irrelevant because we don't care which variable that we assign at which step. So we can start with assigning variable s one or we can start by assigning variable s ten. These steps are not relevant to us. All we care is to reach an assignment that is complete and consistent so we can also use a complete state formulation. So now once we have formulated the CSP solving problem as the search problem then back check in search can be used in order to solve the CSP problem. So in Beijing search, this is the deferred search process using single variable assignment. And the idea is that you go defer a branch of the search tree at each step. You look at one variable that has not been assigned, and then you try to find the values for that variable that does not violate any of the constraints. And then you continue to go down the search tree until if at some point you are able to find a value for the unassigned variables, then you know that that process cannot be continued under that branch. Okay? So there are many nodes that you haven't been explored in the process. And so you go down defer search that one branch but then at some point when you reach a node, when you cannot really find any legitimate assignments for this next variable anymore, then you have to back check to the note above it and then try to find a node assignment of that variables and so on and so forth. And at some point maybe you will need to back check to this one and then have to find the solution under this branch for instance. And so this process is the back checking search to solve using deficit search to solve CSP. Back checking search is the basic uninformed algorithm for CSB and it can solve the n queen problems for n at approximately 25. So if you have the ball having 24 rows and 25 columns then this end queen problem can be solved using simple vegeting search and then above 25 then the search tree becomes too big to be handled by a vegetable search. Now, because this particular technique is uninformed, the question is can our search be better informed? So before we answer the questions, let's look at the bejekin example for the Mac coloring problem for Australia. So we start with the empty assignment so none of the variable has been assigned any of the colors then we choose one state, one region to color it. Let's say that we start with Western Australia. Remember we have three colors red, green and blue. And so in the first branch we call WA by red, in the second branch green and the third graph blue. And then we never search by continue to explore this first branch here and then try to color the next one color the next one Nt. And then because nt is adjacent to WA and therefore nt can only take one of the two color green or blue. And so under this branch now, we have been green for this one, and then the next variable to try is Queensland Q. And let's say that in this particular case, because Queensland is adjacent to Nt and so Nt is already green, so Queensland can only be red or blue.