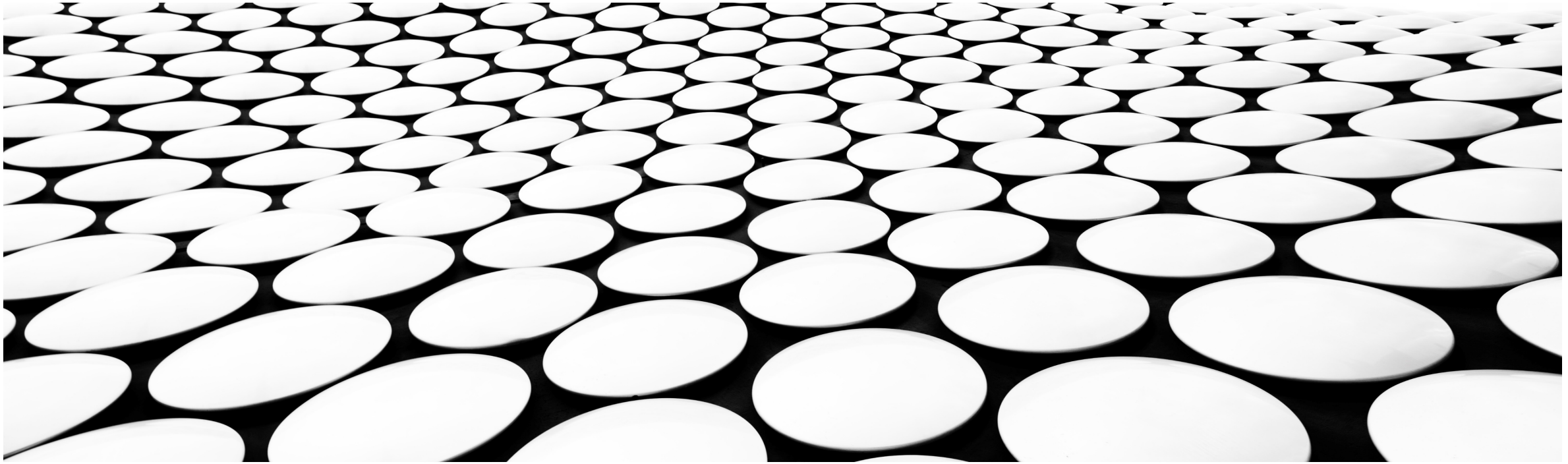
CONSTRAINT SATISFACTION PROBLEM

UNIT III



IN WHICH WE SEE HOW TREATING STATES AS MORE THAN JUST LITTLE BLACK BOXES LEADS TO THE INVENTION OF A RANGE OF POWERFUL NEW SEARCH METHODS AND A DEEPER UNDERSTANDING OF PROBLEM STRUCTURE AND COMPLEXITY.

- In Searching in state space, can be evaluated by domain-specific heuristics and tested to see whether they are goal states.
- From the point of view of the search algorithm, however, each state is atomic, or indivisible—a black box with no internal structure.
- Now describing a way to solve a wide variety of problems more efficiently.
- Using a factored representation for each state: a set of variables, each of which has a value.
- A problem is solved when each variable has a value that satisfies all the constraints on the variable.
- A problem described this way is called **a constraint satisfaction problem, or CSP**.
- Takes advantage of the structure of states and use general-purpose rather than problem-specific heuristics to enable the solution of complex problems.
- The main idea is to eliminate large portions of the search space all at once by identifying variable/value combinations that violate the constraints.

DEFINITION

- A constraint satisfaction problem consists of three components, X,D, and C:

  X is a set of variables, $\{X_1, \ldots, X_n\}$.

  D is a set of domains, $\{D_1, \ldots, D_n\}$, one for each variable.

  C is a set of constraints that specify allowable combinations of values.

- Each domain D_i consists of a set of allowable values, $\{v_1, \ldots, v_k\}$ for variable X_i.
- Each constraint C_i consists of a pair <scope, rel>, where scope is a tuple of variables that participate in the constraint and rel is a relation that defines the values that those variables can take on.
- A relation can be represented as an explicit list of all tuples of values that satisfy the constraint, or as an abstract relation that supports two operations: testing if a tuple is a member of the relation and enumerating the members of the relation.
- For example, if X1 and X2 both have the domain {A,B}, then the constraint saying the two variables must have different values can be written as < (X1,X2), [(A,B), (B,A)]> or as <(X1,X2),X1 ≠ X2>.
- To solve a CSP, we need to define a state space and the notion of a solution.
- Each state in a CSP is defined by an assignment of values to some or all of the variables, {Xi = vi, Xj = vj , . . .}.
- An assignment that does not violate any constraints is called a consistent or legal assignment.
- A complete assignment is one in which every variable is assigned, and a solution to a CSP is a consistent, complete assignment.
- A partial assignment is one that assigns values to only some of the variables

EXAMPLE PROBLEM: MAP COLORING

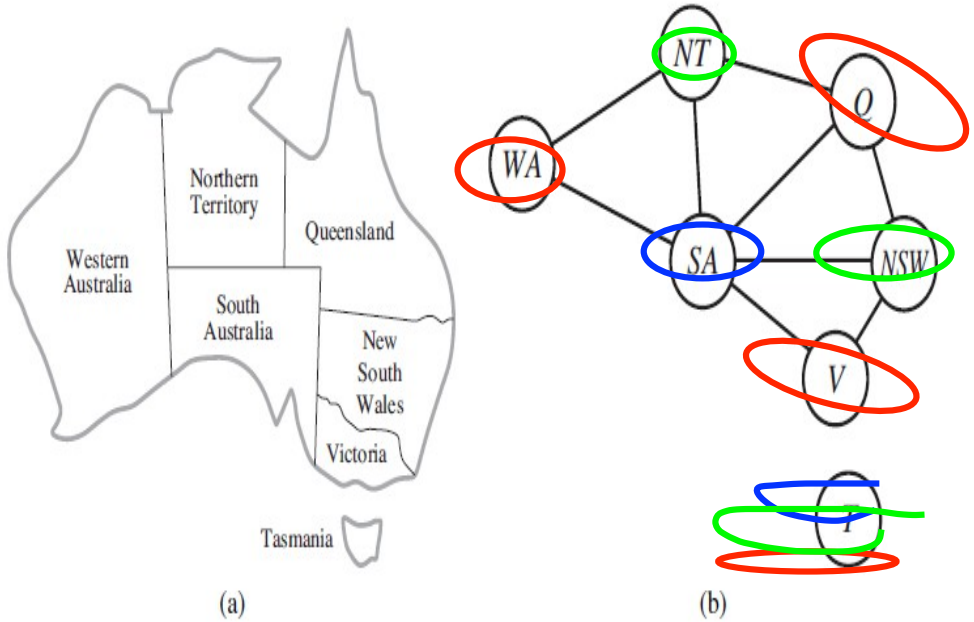
- With colors: red, green, or blue in such a way that no neighboring regions have the same color.
- Define the variables to be the regions X = {WA, NT, Q,NSW, V, SA, T} .
- The domain of each variable is the set D_i = {red , green, blue}.
- The constraints require neighboring regions to have distinct colors.
- Since there are nine places where regions border, there are nine constraints:

C = {SA ≠ WA, SA ≠ NT, SA ≠ Q, SA ≠ NSW, SA ≠ V, WA ≠ NT, NT ≠ Q, Q ≠ NSW, NSW ≠ V } .

- Use abbreviations; SA ≠ WA is a shortcut for <(SA,WA), SA ≠ WA>, where SA≠ WA can be fully enumerated in turn as

{(red , green), (red , blue), (green, red), (green, blue), (blue, red), (blue, green)} .

- There are many possible solutions to this problem, such as {WA=red ,NT =green, Q=red ,NSW =green, V =red ,SA=blue, T =red }.



- The nodes of the graph correspond to variables of the problem, and a link connects any two variables that participate in a constraint.
- **Why CSP?=>** the CSPs yield a natural representation for a wide variety of problems; CSP solvers can be faster than state-space searchers because the CSP solver can quickly eliminate large swatches of the search space.
- For example, once we have chosen {SA=blue} in the Australia problem, we can conclude that none of the five neighboring variables can take on the value blue.
- Without taking advantage of constraint propagation, a search procedure would have to consider 3^5 =243 assignments for the five neighboring variables; with constraint propagation we never have to consider blue as a value, so we have only 2^5 =32 assignments to look at, a reduction of 87%.
- In regular state-space search we can only ask: is this specific state a goal? No? What about this one? With CSPs, once we find out that a partial assignment is not a solution, we can immediately discard further refinements of the partial assignment. Furthermore, we can see why the assignment is not a solution—we see which variables violate a constraint—so we can focus attention on the variables that matter. As a result, many problems that are intractable for regular state-space search can be solved quickly when formulated as a CSP.

JOB SCHEDULING PROBELM

- We consider a small part of the car assembly, consisting of 15 tasks:

  install axles (front and back), affix all four wheels (right and left, front and back), tighten nuts for each wheel, affix hubcaps, and inspect the final assembly.

- We can represent the tasks with **15 variables:**

  X = {AxleF , AxleB,WheelRF ,WheelLF ,WheelRB,WheelLB, NutsRF , NutsLF , NutsRB, NutsLB, CapRF , CapLF , CapRB, CapLB, Inspect} .

- The **value** of each variable is the time that the task starts.

- **Constraints:** Precedence constraints, Whenever a task T1 must occur before task T2 PRECEDENCE , and task T1 takes duration d1 to complete,

  An arithmetic constraint of the form: $T1 + d1 \leq T2$.

- Now the axles have to be in place before the wheels are put on, and it takes 10 minutes to install an axle, so we write

  $AxleF + 10 \leq WheelRF$; $AxleF + 10 \leq WheelLF$;

  $AxleB + 10 \leq WheelRB$; $AxleB + 10 \leq WheelLB$.

- Next we say that, for each wheel, we must affix the wheel (which takes 1 minute), then tighten the nuts (2 minutes), and finally attach the hubcap (1 minute, but not represented yet):

  $WheelRF + 1 \leq NutsRF$; $NutsRF + 2 \leq CapRF$;

  $WheelLF + 1 \leq NutsLF$; $NutsLF + 2 \leq CapLF$;

  $WheelRB + 1 \leq NutsRB$; $NutsRB + 2 \leq CapRB$;

  $WheelLB + 1 \leq NutsLB$; $NutsLB + 2 \leq CapLB$.

- Suppose we have four workers to install wheels, but they have to share one tool that helps put the axle in place. We need a disjunctive constraint to say that AxleF and AxleB must not overlap in time; either one comes first or the other does: (AxleF + 10 ≤ AxleB) or (AxleB + 10 ≤ AxleF) .
- Complicated constraint, combining arithmetic and logic. But it still reduces to a set of pairs of values that AxleF and AxleF can take on.
- We also need to assert that the inspection comes last and takes 3 minutes.
- For every variable except Inspect we add a constraint of the form X +dX ≤ Inspect .
- Finally, suppose there is a requirement to get the whole assembly done in 30 minutes. We can achieve that by limiting the domain of all variables:

  Di = {1, 2, 3, . . . , 27} .
- This particular problem is trivial to solve, but CSPs have been applied to job-shop scheduling problems like this with thousands of variables. In some cases, there are complicated constraints that are difficult to specify in the CSP formalism, and more advanced planning techniques are used.

DEFINITIONS:

- Discrete, finite Domains:
- Infinite Domains:
- Linear Constraints
- Non Linear Constraints
- Continuous Domains: Constraint satisfaction problems with **continuous domains** are common in the real world and are widely studied in the field of operations research. E.g Hubble Space Telescope
- Unary Constraints: Single variable
- Binary constraints: Two variable
- Global constraints: an arbitrary number of variables, *Alldiff*
- *In CSPs there is a choice: an algorithm can search (choose a new variable assignment from several possibilities) or do a specific type of inference called* ***constraint propagation****:*
- *using the constraints to reduce the number of legal values for a variable, which in turn can reduce the legal values for another variable, and so on.*
- Constraint propagation may be intertwined with search, or it may be done as a preprocessing step, before search starts. Sometimes this preprocessing can solve the whole problem, so no search is required at all.

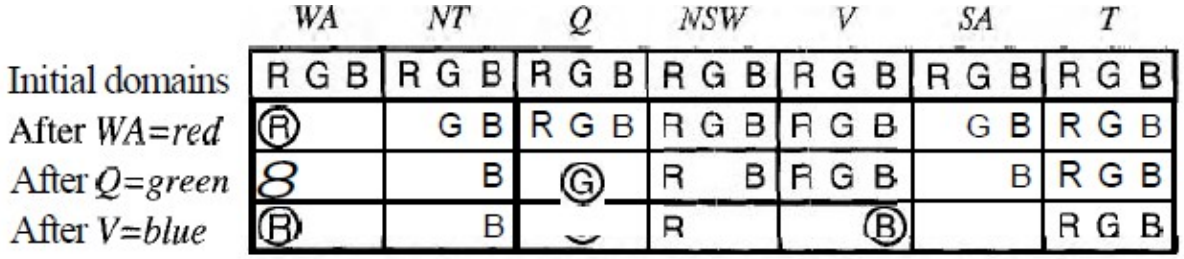
LOCAL CONSISTENCY

- If we treat each variable as a node in a graph and each binary constraint as an arc, then the process of enforcing local consistency in each part of the graph causes inconsistent values to be eliminated throughout the graph.
- There are different types of local consistency:
- **Node consistency:** single variable (corresponding to a node in the CSP network) is node-consistent if all the values in the variable's domain satisfy the variable's unary constraints.
- **Arc consistency:** A variable in a CSP is **arc-consistent** if every value in its domain satisfies the variable's binary constraints. More formally, Xi is arc-consistent with respect to another variable Xj if for every value in the current domain Di there is some value in the domain Dj that satisfies the binary constraint on the arc (Xi,Xj). A network is arc-consistent if every variable is arc consistent with every other variable. For example, consider the constraint $Y = X^2$ where the domain of both X and Y is the set of digits.
- **Path consistency: Path consistency** tightens the binary constraints by using implicit constraints that are inferred by looking at triples of variables. A two-variable set {Xi,Xj} is path consistent with respect to a third variable Xm if, for every assignment {Xi = a,Xj = b} consistent with the constraints on {Xi,Xj}, there is an assignment to Xm that satisfies the constraints on {Xi,Xm} and {Xm,Xj}. This is called path consistency because one can think of it as looking at a path from Xi to Xj with Xm in the middle.
- *K***-consistency:** Stronger forms of propagation can be defined with the notion of k**-consistency**. A CSP is k-consistent if, for any set of $k - 1$ variables and for any consistent assignment to those variables, a consistent value can always be assigned to any kth variable. 1-consistency says that, given the empty set, we can make any set of one variable consistent: this is what we called node consistency. 2-consistency is the same as arc consistency. For binary constraint networks, 3-consistency is the same as path consistency. A CSP is **strongly** k**-consistent** if it is k-consistent and is also $(k - 1)$-consistent, $(k - 2)$-consistent, . . . all the way down to 1-consistent.

BACKTRACKING SEARCH FOR CSPS

- Sudoku problems are designed to be solved by inference over constraints. But many other CSPs cannot be solved by inference alone; there comes a time when we must search for a solution.
- In this section we look at backtracking search algorithms that work on partial assignments;
- A state = partial assignment, and an action = adding var = value to the assignment.
- But for a CSP with n variables of domain size d, we quickly notice something terrible: the branching factor at the top level is nd because any of d values can be assigned to any of n variables. At
- the next level, the branching factor is (n − 1)d, and so on for n levels. We generate a tree
- with n! · dn leaves, even though there are only dn possible complete assignments!

FORWARD CHECKING



	WA	NT	Q	NSW	V	SA	T
Initial domains	R G B	R G B	R G B	R G B	R G B	R G B	R G B
After *WA=red*	(R)	G B	R G B	R G B	R G B	G B	R G B
After *Q=green*	8	B	(G)	R B	R G B	B	R G B
After *V=blue*	(R)	B	‿	R	(B)		R G B

- One way to make better use of constraints during search is called **forward checking.**
- Whenever **a** variable X is assigned, the forward checking process looks at each unassigned variable Y that is connected to X by a constraint and deletes from Y's domain any value that is inconsistent with the value chosen for X.
- Figure shows the progress of a map-coloring search with forward checking. There are two important points to notice about this example.
- First, notice that after assigning WA = red and Q = green, the domains of NT and SA are reduced to a single value; we have eliminated branching on these variables altogether by propagating information from WA and Q.
- The MRV heuristic, which is an obvious partner for forward checking, would automatically select SA and NT next. (Indeed, we can view forward checking as an efficient way to incrementally compute the information that the MRV heuristic needs to do its job.) A second point to notice is that, after V = *blue*, the domain of SA is empty. Hence, forward checking has detected that the partial assignment
- { WA = *red*, Q = *green*, *V* = *blue)* is inconsistent with the constraints of the problem, and the algorithm will therefore backtrack immediately

CONSTRAINT PROPAGATION

- Although forward checking detects many inconsistencies, it does not detect all of them. For example, consider the third row of Figure 5.6. It shows that when WA is *red* and Q is *green,* both NT and SA are forced to be blue. But they are ,adjacent and so cannot have the same value.
- Forward checking does not detect this as an inconsistency, because it does not look far CPROONPSATGRAATINITO N enough ahead.
- **Constraint propagation** is the general term for propagating the implications of a constraint on one variable onto other variables; xn this case we need to propagate from WA and Q onto NT and SA, (as was done by forward checking) and then onto the constraint between NT and SA to detect the inconsistency.
- And we want to do this fast: it is no good reducing the amount of search if we spend more time propagating constraints than we would have spent doing a simple search.
- ARC CONSISTENCY The idea of **arc consistency** provides a fast method of constraint propagation that is substantially stronger than forward checking. Here, "arc" refers to a *directed* arc in the constraint graph, such as the arc from SA to NS W. Given the current domains of SA and NS W, the arc is consistent if, for *every* value x of SA, there is *some* value y of NS W that is consistent with x.
- In the third row of Figure 5.6, the current domains of SA and NSW are *{blue)* and *{red, blue)* respectively. For SA = *blue,* there is a consistent assignment for NSW, namely, NSW = *red;* therefore, the arc from SA to I1JS'W is consistent. On the other hand, the reverse arc from NS W to SA is not consistent: for Ithe assignment NS W = *blue,* there is no consistent assignment for SA. The arc can be made consistent by deleting the value *blue* from the domain of *NS* W.
- We can also apply arc consistency to the arc from SA to NT at the same stage in the search process. The third row of the table in Figure 5.6 shows that both variables have the domain *{blue).* The result is that *blue* must be deleted from the domain of SA, leaving the domain empty.
- Thus, applying arc consistency has resulted ira early detection of an inconsis-