Artificial Intelligence

Dr. Qaiser Abbas
Department of Computer Science & IT
University of Sargodha

1. CONSTRAINT SATISFACTION PROBLEMS

- We use 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 (CSP).

1.1. DEFINING CONSTRAINT SATISFACTION PROBLEMS

- A constraint satisfaction problem consists of three components, X, D, and C :
  - X is a set of variables, $\{X_1,...,X_n\}$.
  - D is a set of domains, $\{D_1, . . . , D_n\}$, one for each variable.
  - C is a set of constraints that specify allowable combinations of values.

  - ⟨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.
    - For example, if X1 and X2 both have the domain values {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⟩.

1.1. DEFINING CONSTRAINT SATISFACTION PROBLEMS

- 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, $\{X_i = v_i, X_j = v_j , . . .\}$.
- 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.
- A **partial assignment** is one that assigns values to only some of the variables.

1.1.1 Example problem: Map coloring

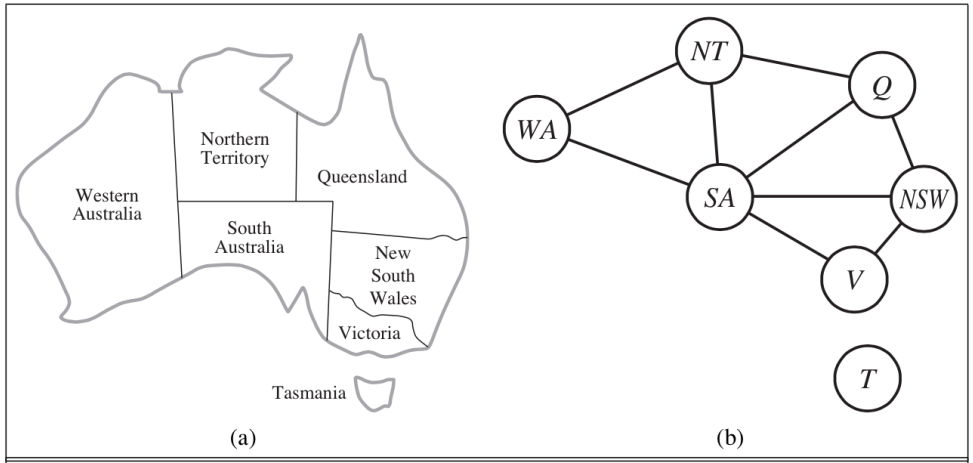


Figure 6.1 (a) The principal states and territories of Australia. Coloring this map can be viewed as a constraint satisfaction problem (CSP). The goal is to assign colors to each region so that no neighboring regions have the same color. (b) The map-coloring problem represented as a constraint graph.

- Variable X = {WA,NT,Q,NSW,V,SA,T} .
- Domain D = {red , green , blue }.
- Constraint C = {neighboring regions to have distinct colors}

1.1.1 Example problem: Map coloring

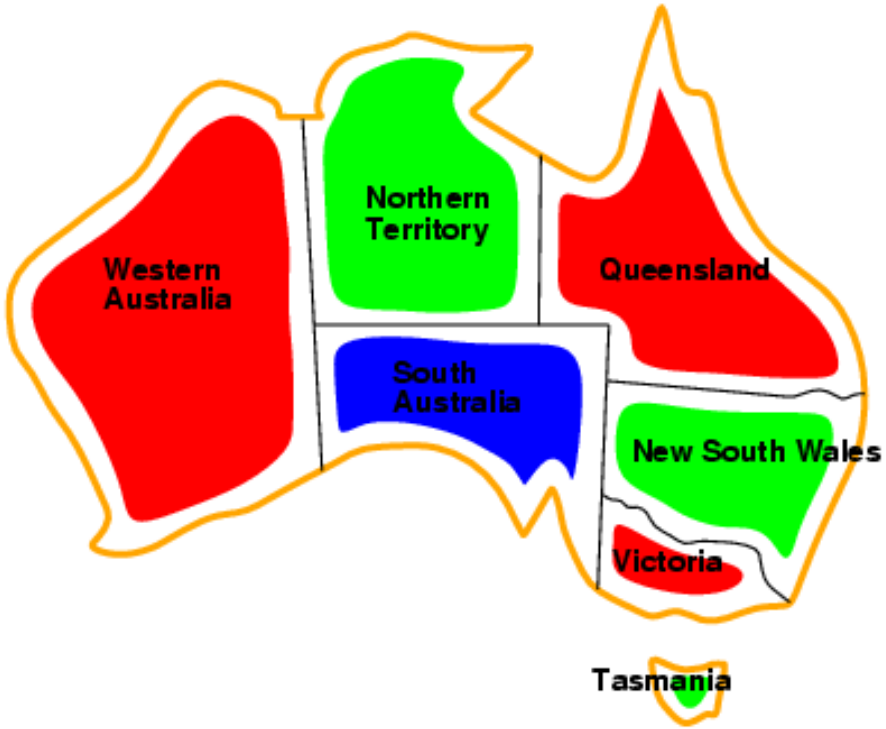
- It can be helpful to visualize a CSP as a constraint graph, as shown in Figure 6.1(b). The nodes of the graph correspond to variables of the problem, and a link connects any two variables that participates in a constraint.

- Since there are nine places where regions border (See Fig 6.1), there are nine constraints as follows:
  C = {SA≠ WA, SA≠ NT, SA≠ Q, SA≠ NSW, SA≠ V, WA≠NT, NT≠Q, Q≠NSW, NSW≠V}.

- SA≠ WA is a shortcut of its formal representation (scope, rel) as ⟨(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 }, can be figuratively on next slide.

1.1.1 Example problem: Map coloring



- For {SA = blue} in the Australia problem, 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%.

1.1.2. Example problem: Job-shop scheduling

- Whenever a task T_1 must occur before task T_2 , and task T_1 takes duration d_1 to complete, we add an arithmetic constraint of the form $T_1 + d_1 \leq T_2$. (**precedence constraints**)

- We need a **disjunctive constraint** to say that $Axle_F$ and $Axle_B$ must not overlap in time; either one comes first or the other does:

  $(Axle_F + 10 \leq Axle_B)$ **or** $(Axle_B + 10 \leq Axle_F)$.

- Read it yourself

1.1.3 Variations on the CSP formalism

- **Discrete variables**
  - finite domains:
    - *n* variables, domain size *d* → $O(d^n)$ complete assignments e.g., Map coloring
  - infinite domains:
    - integers, strings, etc. e.g., a constraint language like $T_1 + d_1 \leq T_2$

- **Unary constraint**, which restricts the value of a single variable. For example, in the map-coloring problem it could be the case that South Australians won't tolerate the color green; we can express that with the unary constraint ⟨(SA),SA ≠ green⟩

1.1.3 Variations on the CSP formalism

- A **binary constraint** relates two variables. For example, SA≠ NSW is a binary constraint as in Figure 6.1(b).

- We can also describe **higher-order constraints**, such as asserting that the value of Y is between X and Z, with the ternary constraint Between(X, Y, Z).

- A constraint involving an arbitrary number of variables is called a **global constraint**. One of the most common global constraints is *Alldiff*, which says that all of the variables involved in the constraint must have different values. Its example is **cryptarithmetic** puzzles given on the next slide.

1.1.3 Variations on the CSP formalism

- **Crypto-Arithmetic Problem:**
  - Crypt-Arithmetic Problems are substitution problems where letters representing a mathematical operation are replaced by unique digits.
    Like : **P L A Y S + W E L L = B E T T E R**

  - Where each unique alphabet represents a unique digit from among 0 to 9. So, if the solution to this puzzle is to be found, it would be (after a long computation) :

    9 7 4 2 6 + 8 0 7 7 = 1 0 5 5 0 3

1.1.3 Variations on the CSP formalism

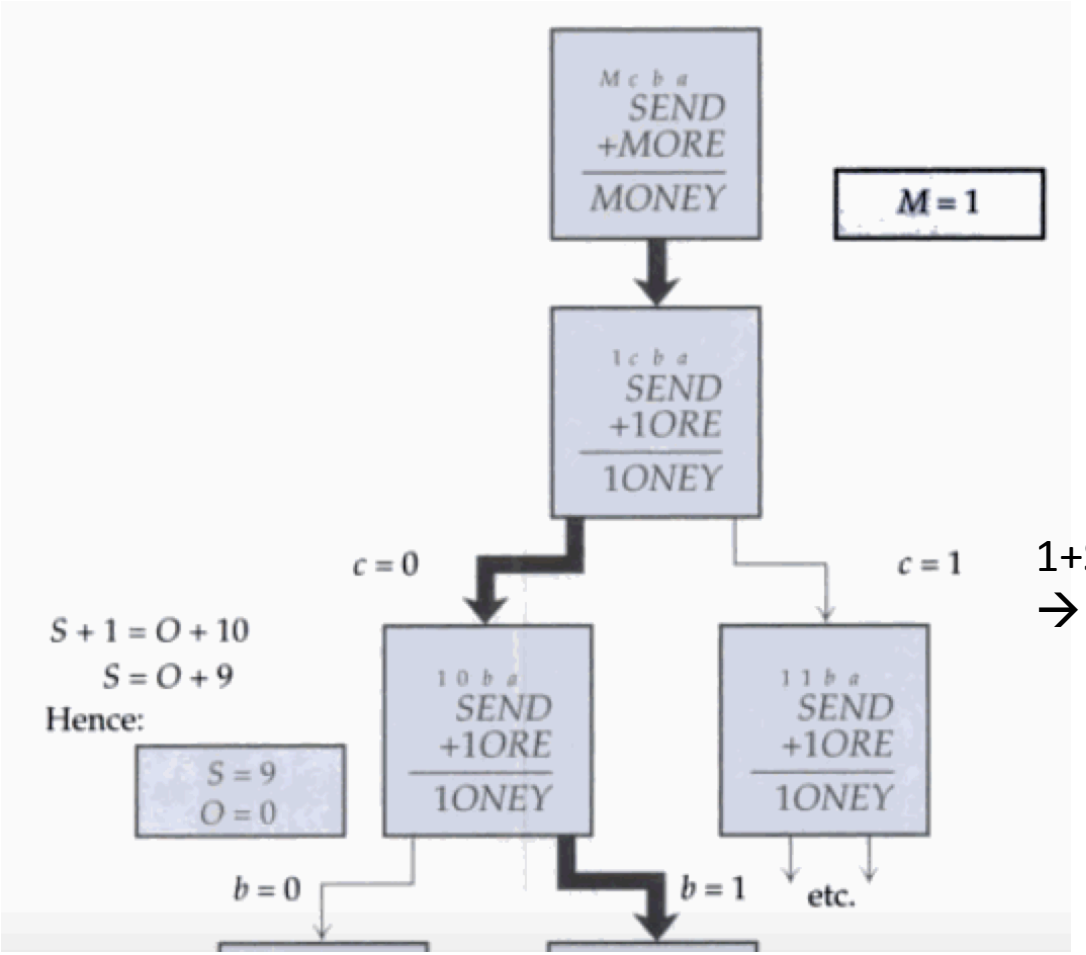
- **Cryptarithmetic Example:**
  - The basic rules are :
    - Each unique digit (from 0 to 9) must be replaced by a unique character.
    - The number so formed cannot start with a ZERO.
    - SEND + MORE = MONEY
  - **Solution**:
    - Total number of candidate solution: $10^8 \times 2^4 = 160{,}000{,}000{,}0$ for 10 digits, 8 letters, 2 carry digits and 4 carries involved, however, if M =1, then 800,000,000

$$\begin{array}{r} {\scriptstyle M\ c\ b\ a} \\ SEND \\ +\ MORE \\ \hline MONEY \end{array}$$

$$D + E = 10a + Y \qquad b + E + O = 10c + N$$

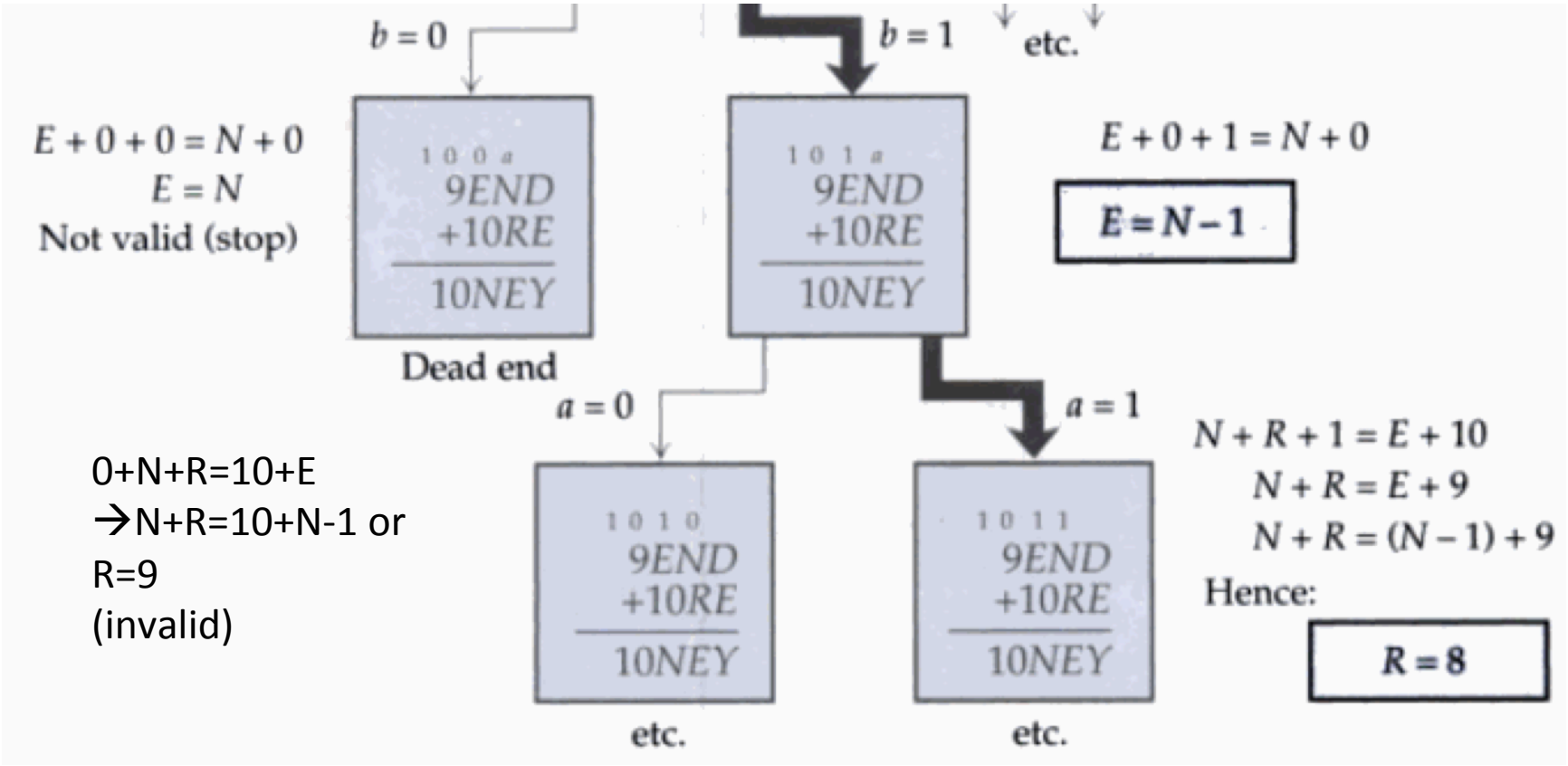
$$a + N + R = 10b + E \qquad c + S + M = 10M + O$$

1.1.3 Variations on the CSP formalism



1+S+1 = 10 + O
→ S = 8 + O

1.1.3 Variations on the CSP formalism



1.1.3 Variations on the CSP formalism

```
1   0 1 1
    9 E N D
  + 1 0 8 E
----------------
1   0 N E Y
```

- When R=8,
  - Now, if E = 5, then 5 = N -1 → **N =6**
  - Putting these values, we have
  - Now, think some number for D, which satisfies all the constraints e.g. if D = ~~1,2,3,4,5,6~~, 7, then y becomes 2.
  - So, S=9, E=5, N=6, D=7, M=1, O=0, R=8, Y =2

```
1   0 1 1
    9 5 6 D
  + 1 0 8 5
----------------
1   0 6 5 Y
```

```
1   0 1 1
    9 5 6 7
  + 1 0 8 5
----------------
1   0 6 5 2
```

1.2 BACKTRACKING SEARCH FOR CSPs

- **CSP Standard search formulation (incremental)**
  - Let's start with the straightforward approach, then fix it.
  - States are defined by the values assigned so far:
    - Initial state: the empty assignment { }
    - Successor function: assign a value to an unassigned variable that must be legal assignment. This becomes fail if no legal assignments.
    - Goal test: the current assignment is complete
  - This is the same for all CSPs and it uses depth first search.
  - Variable assignments in all CSPs are commutative, i.e., [WA = red then NT = green] same as [NT = green then WA = red]

1.2 BACKTRACKING SEARCH FOR CSPs

- Backtracking search is used for a **depth-first search** that chooses values for one variable at a time and backtracks when a variable has no legal values left to assign.

- The algorithm is shown in Figure next. It repeatedly chooses an unassigned variable, and then tries all values in the domain of that variable in turn, trying to find a solution.

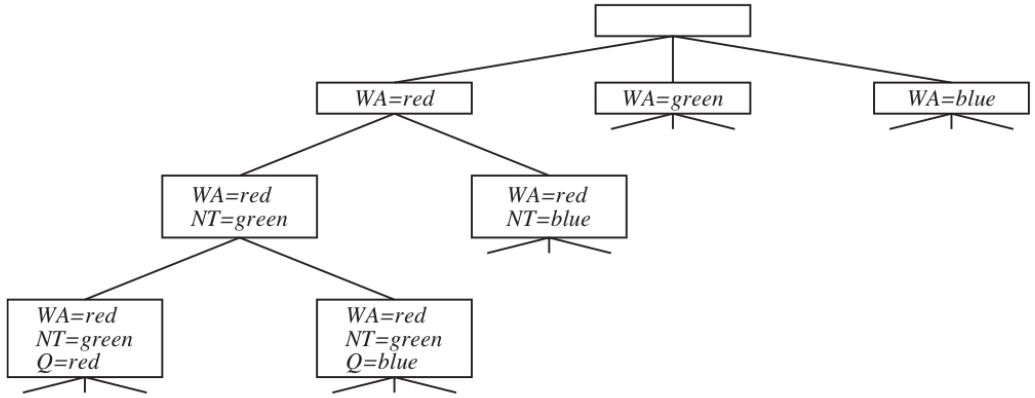
- If an inconsistency is detected, then BACKTRACK returns failure, causing the previous call to try another value.

- Part of the search tree for the Australia problem is shown in Figure next, where we have assigned variables in order of WA, NT , Q, ...

1.2 BACKTRACKING SEARCH FOR CSPs

```
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
  return BACKTRACK({ }, csp)

function BACKTRACK(assignment, csp) returns a solution, or failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
    if value is consistent with assignment then
      add {var = value} to assignment
      inferences ← INFERENCE(csp, var, value)
      if inferences ≠ failure then
        add inferences to assignment
        result ← BACKTRACK(assignment, csp)
        if result ≠ failure then
          return result
    remove {var = value} and inferences from assignment
  return failure
```

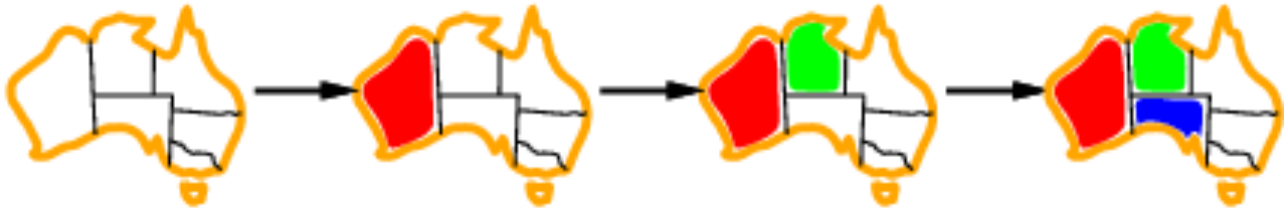


1.2 BACKTRACKING SEARCH FOR CSPs

- **Improving backtracking efficiency**
  - Which variable should be assigned next (SELECT-UNASSIGNED-VARIABLE), and in what order should its values be tried (ORDER-DOMAIN-VALUES)?
  - What inferences should be performed at each step in the search (INFERENCE)?
    - Can we detect inevitable(unavoidable) failure early?
  - When the search arrives at an assignment that violates a constraint, can the search avoid repeating this failure?

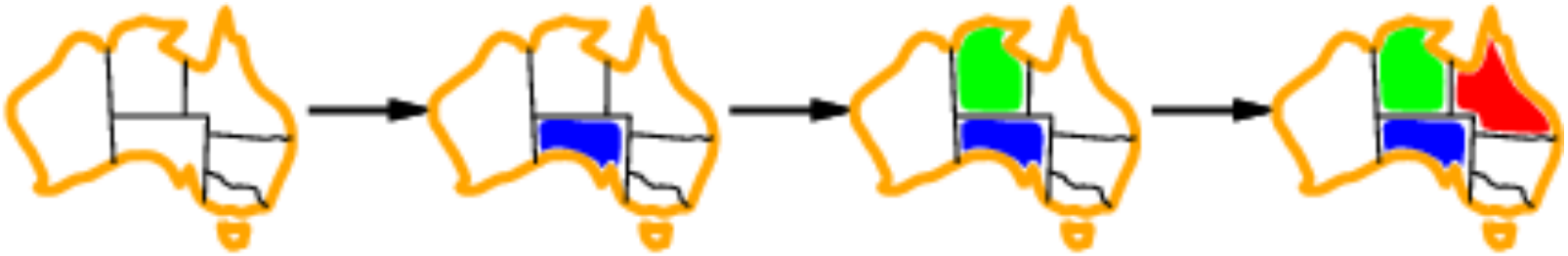
1.2.1 Variable and value ordering

- **Minimum- remaining-values (MRV) heuristic**
  - The simplest strategy for SELECT-UNASSIGNED-VARIABLE is to choose the next unassigned variable in order, {X1, X2, . . .}. This static variable ordering seldom results in the most efficient search.
  - For example, after the assignments for WA = red and NT = green in Figure, there is only one possible value for SA, so it makes sense to assign SA = blue next rather than assigning Q.
  - In fact, after SA is assigned, the choices for Q, NSW , and V are all forced. This intuitive idea—choosing the variable with the fewest "legal" values—is called the **minimum-remaining-values** (MRV) heuristic.
  - It also has been called the "most constrained variable" or "fail-first" heuristic, the latter because it picks a variable that is most likely to cause a failure soon, thereby pruning the search tree.



1.2.1 Variable and value ordering

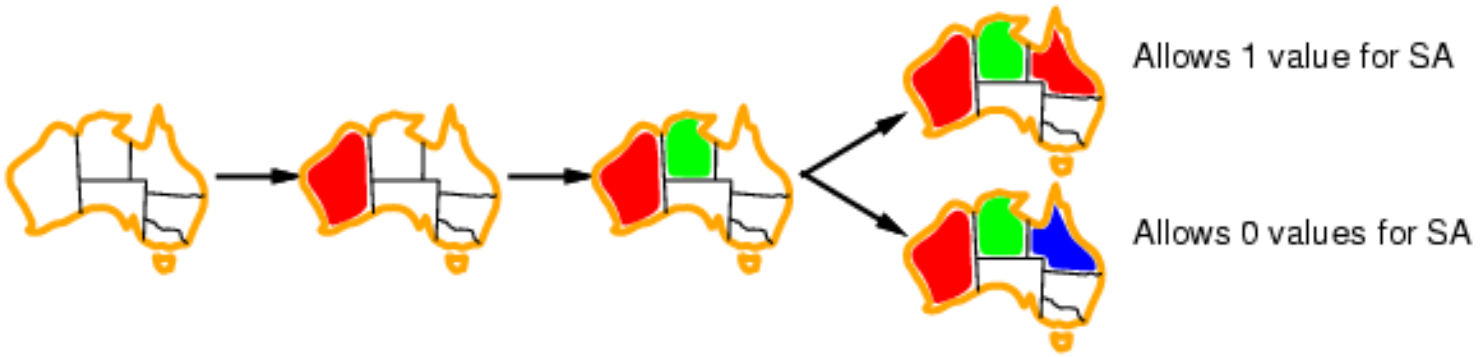
- **Degree heuristic:**
  - The MRV heuristic doesn't help at all in choosing the first region to color in Australia, because initially every region has three legal colors.
  - In this case, the **degree heuristic** comes in handy. It attempts to reduce the branching factor on future choices by selecting the variable that is involved in the largest number of constraints on other unassigned variables.
  - In Figure 6.1, SA is the variable with highest degree, 5; the other variables have degree 2 or 3, except for T , which has degree 0.
  - In fact, once SA is chosen, applying the degree heuristic solves the problem without any false steps—you can choose *any* consistent color at each choice point and still arrive at a solution with no backtracking. The MRV heuristic is usually a more powerful guide, but the degree heuristic can be useful as a tie-breaker.



1.2.1 Variable and value ordering

- **Least-constraining-value**
  - Given a variable, choose the least constraining value. For example as follows.

  - we have generated the partial assignment with WA = red and NT = green and that our next choice is for Q. Blue would be a bad choice because it eliminates the last legal value left for Q's neighbor, SA. The least-constraining-value heuristic therefore prefers red to blue. In general, the heuristic is trying to leave the maximum flexibility for subsequent variable assignments.

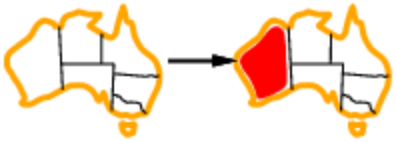


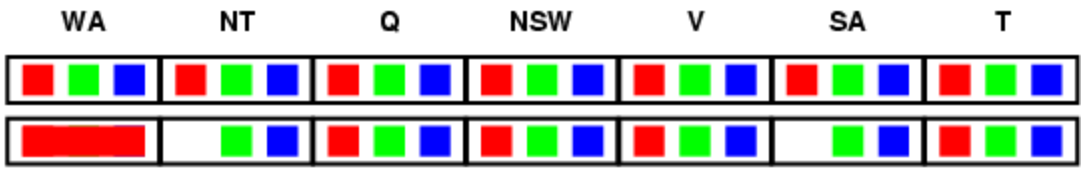
1.2.2 Interleaving search and inference

- **Forward Checking**
  - One of the simplest forms of inference is called **forward checking**.
  - Whenever a variable X is assigned, the forward-checking process establishes arc consistency for it:
    - for each unassigned variable Y that is connected to X by a constraint, delete from Y 's domain any value that is inconsistent with the value chosen for X.
  - Because forward checking only does arc consistency inferences, there is no reason to do forward checking if we have already done arc consistency as a preprocessing step.

1.2.2 Interleaving search and inference

- **Forward Checking**
  - One of the simplest forms of inference is called **forward checking**.
  - Whenever a variable X is assigned, the forward-checking process establishes arc consistency for it:
    - for each unassigned variable Y that is connected to X by a constraint, delete from Y 's domain any value that is inconsistent with the value chosen for X.
  - Because forward checking only does arc consistency inferences, there is no reason to do forward checking if we have already done arc consistency as a preprocessing step.



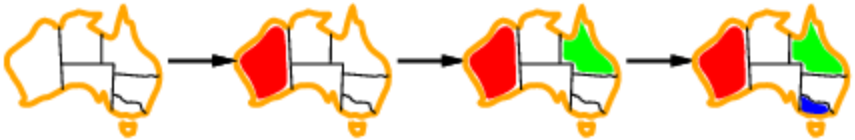


1.2.2 Interleaving search and inference

- **Forward Checking**
  - One of the simplest forms of inference is called **forward checking**.
  - Whenever a variable X is assigned, the forward-checking process establishes arc consistency for it:
    - for each unassigned variable Y that is connected to X by a constraint, delete from Y 's domain any value that is inconsistent with the value chosen for X.
  - Because forward checking only does arc consistency inferences, there is no reason to do forward checking if we have already done arc consistency as a preprocessing step.

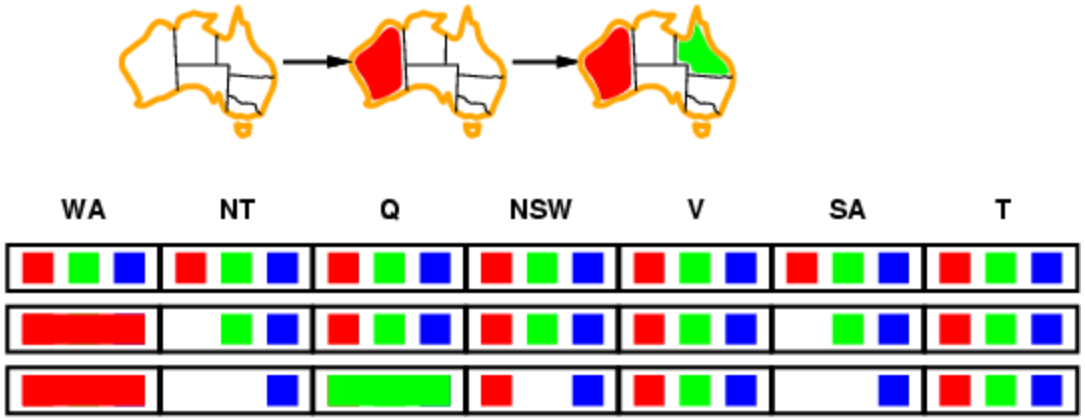
1.2.2 Interleaving search and inference

- **Forward Checking**
  - One of the simplest forms of inference is called **forward checking**.
  - Whenever a variable X is assigned, the forward-checking process establishes arc consistency for it:
    - for each unassigned variable Y that is connected to X by a constraint, delete from Y 's domain any value that is inconsistent with the value chosen for X.
  - Because forward checking only does arc consistency inferences, there is no reason to do forward checking if we have already done arc consistency as a preprocessing step.



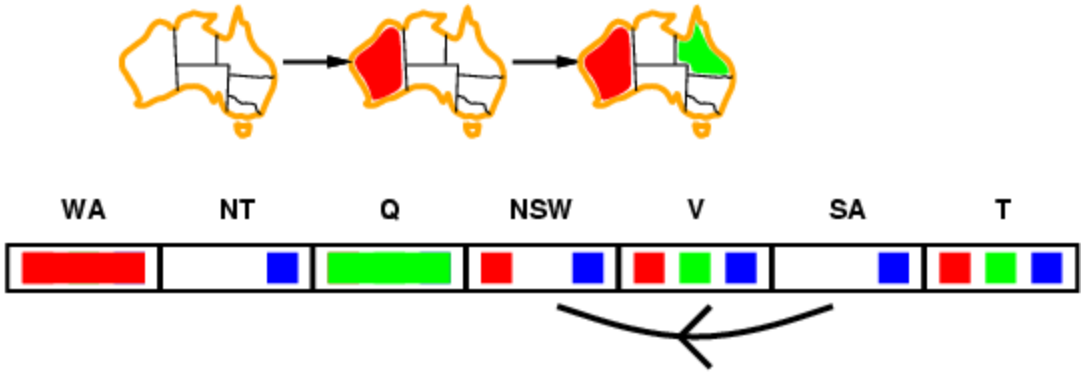
1.2.2 Interleaving search and inference

- **Drawback in Forward Checking**
  - Although forward checking detects many inconsistencies, it does not detect all of them. The problem is that it makes the current variable arc-consistent, but doesn't look ahead and make all the other variables arc-consistent.
  - For example, consider the Figure. It shows that when WA is red and Q is green , both NT and SA are forced to be blue.
  - Forward checking does not look far enough ahead to notice that this is an inconsistency: NT and SA are adjacent and so cannot have the same value.



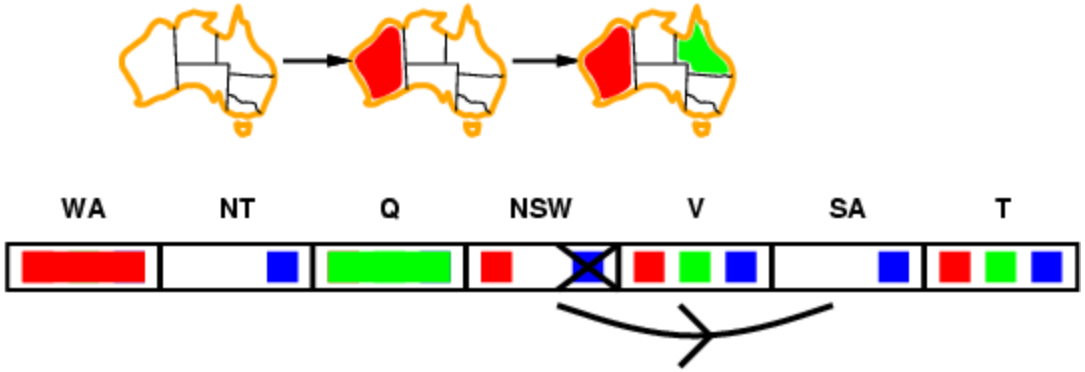
1.2.2 Interleaving search and inference

- **Arc Consistency**
  - Simplest form of propagation makes each arc consistent.
  - $X \rightarrow Y$ is consistent iff for every value x of X there is some allowed y



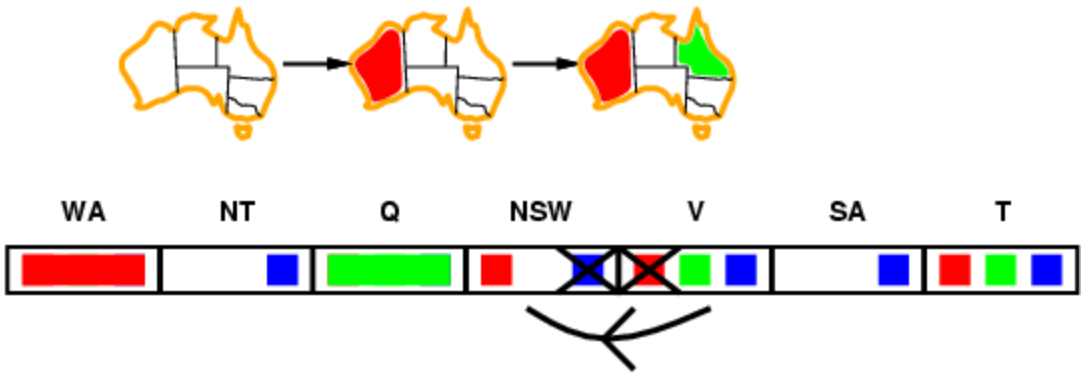
1.2.2 Interleaving search and inference

- **Arc Consistency**
  - Simplest form of propagation makes each arc consistent.
  - *X* → *Y* is consistent iff for every value *x* of *X* there is some allowed *y*



1.2.2 Interleaving search and inference

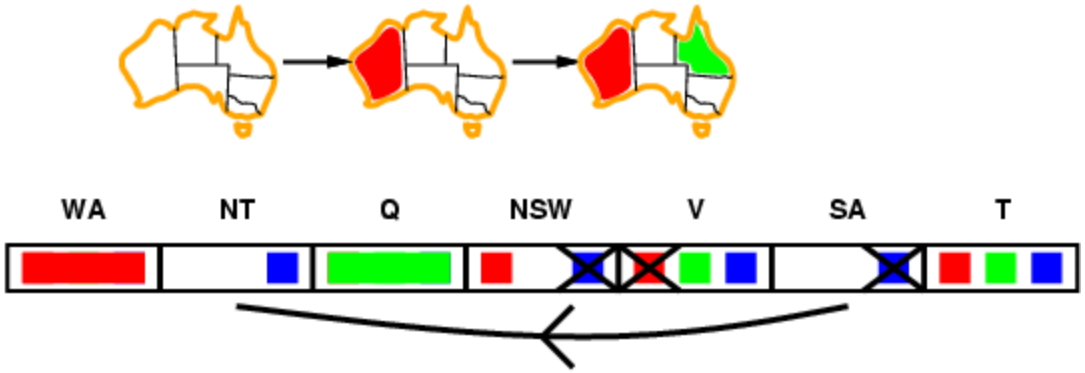
- **Arc Consistency**
  - Simplest form of propagation makes each arc consistent.
  - $X \rightarrow Y$ is consistent iff for every value x of X there is some allowed y



  - If X loses a value, neighbors of X need to be rechecked

1.2.2 Interleaving search and inference

- **Arc Consistency**
  - Simplest form of propagation makes each arc consistent.
  - $X \rightarrow Y$ is consistent iff for every value x of X there is some allowed y



  - If X loses a value, neighbors of X need to be rechecked
  - Arc consistency detects failure earlier than forward checking
  - Can be run as a preprocessor or after each assignment

1.2.3 Intelligent backtracking: Looking backward

- The BACKTRACKING-SEARCH algorithm has a very simple policy for what to do when a branch of the search fails: back up to the preceding variable and try a different value for it. This is called **chronological** (order as variables occurred) **backtracking**.

- Consider what happens when we apply simple backtracking in Figure 6.1 with a fixed variable ordering Q, NSW , V , T , SA, WA, NT . Suppose we have generated the partial assignment {Q=red,NSW =green,V =blue,T =red}. When we try the next variable, SA, we see that every value violates a constraint. We back up to T and try a new color for Tasmania! Obviously this is silly—recoloring Tasmania cannot possibly resolve the problem with South Australia.

- A more intelligent approach to backtracking is to backtrack to a variable that might fix the problem—a variable that was responsible for making one of the possible values of SA impossible. To do this, we will keep track of a set of assignments that are in conflict with some value for SA. The set (in this case {Q=red,NSW =green,V =blue,}), is called the **conflict set** for SA. The **backjumping** method backtracks to the *most recent* assignment in the conflict set; in this case, backjumping would jump over Tasmania and try a new value for V . A backjumping algorithm that uses conflict sets defined in this way is called conflict-directed backjumping.

1.3 LOCAL SEARCH FOR CSPs

- Read it yourself

Additional Readings

- THE STRUCTURE OF PROBLEMS (See Section 6.5)
- CONSTRAINT PROPAGATION: INFERENCE IN CSPs (See Section 6.2)

Lab Project No. 6

- **Experiment** the Graph Coloring CSP or Cryptarithmetic Puzzle
  - Go to the following for Graph Coloring at https://www.geeksforgeeks.org/backttracking-set-5-m-coloring-problem/
  - Or visit the following link for Crypt arithmetic Puzzle at

https://www.codeproject.com/Articles/176768/Cryptarithmetic

  - C/C++, Java and python source code is available, download it, configure & execute it and submit a report accordingly.