CONSTRAINT SATISFACTION PROBLEMS

Course Code: **CSC4226** Course Title: **Artificial Intelligence and Expert System**

Dept. of Computer Science
Faculty of Science and Technology

Many slides from Dan Klein

Lecture No:	**Eight (7)**	**Week No:**	**Eight (8)**	**Semester:**	
Lecturer:	*Shaikat Das Joy*			*skdas@aiub.edu*	

Lecture Outline

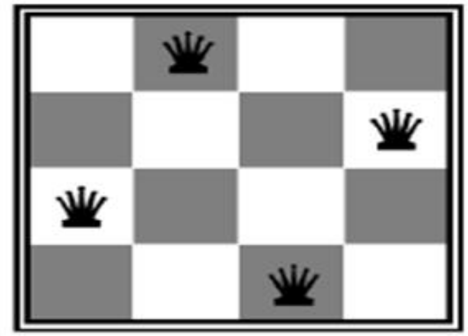
1. Defining CSP
2. Varieties of CSPs
3. Backtracking Search for CSPs
4. Forward Checking
5. Constraint Propagation
6. Heuristics for CSP
7. Problem Structure

What is Search For?

- Models of the world: single agents, deterministic actions, fully observed state, discrete state space
- Planning: sequences of actions
  - The path to the goal is the important thing
  - Paths have various costs, depths
  - Heuristics to guide, fringe to keep backups
- Identification: assignments to variables
  - The goal itself is important, not the path
  - All paths at the same depth (for some formulations)
  - CSPs are specialized for identification problems

Constraint Satisfaction Problems

- Standard search problems:
  - State is a "black box": arbitrary data structure
  - Goal test: any function over states
  - Successor function can be anything



- Constraint satisfaction problems (CSPs):
  - A special subset of search problems
  - State is defined by variables X_i with values from a domain D (sometimes D depends on i)
  - Goal test is a set of constraints specifying allowable combinations of values for subsets of variables

- Simple example of a *formal representation language*

- Allows useful general-purpose algorithms with more power than standard search algorithms

Example of CSPs

Some poplar puzzles like, the Latin Square, the Eight Queens, and Sudoku are stated below.

- **Latin Square Problem :** How can one fill an $n \times n$ table with n different symbols such that each symbol occurs exactly once in each row and each column ?

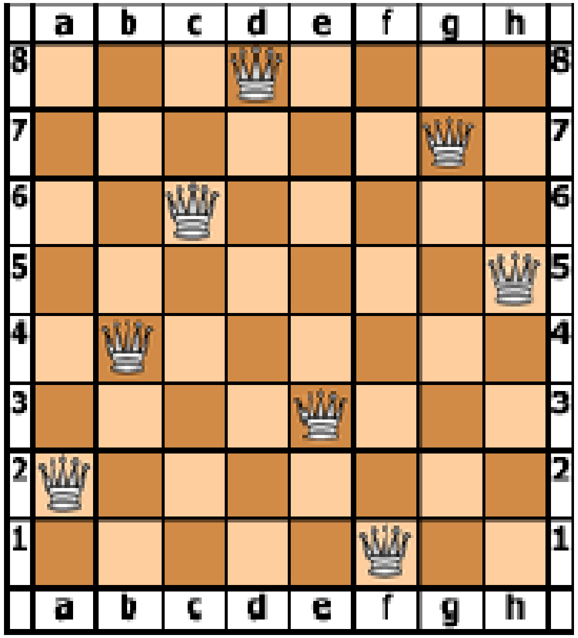
  Solutions : The Latin squares for $n = 1, 2, 3$ and 4 are :

$$\begin{pmatrix} 1 \end{pmatrix} \quad \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix} \quad \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \end{pmatrix} \quad \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 1 \\ 3 & 4 & 1 & 2 \\ 4 & 1 & 2 & 3 \end{pmatrix}$$

Example of CSPs

- **Eight Queens Puzzle Problem :** How can one put 8 queens on a (8 x 8) chess board such that no queen can attack any other queen ?

Solutions: The puzzle has 92 distinct solutions. If rotations and reflections of the board are counted as one, the puzzle has 12 unique solutions.



Unique solution 1

Example CSP: Map Coloring

- Variables: $WA,\ NT,\ Q,\ NSW,\ V,\ SA,\ T$
- Domain: $D = \{red, green, blue\}$
- Constraints: adjacent regions must have different colors

$$WA \neq NT$$

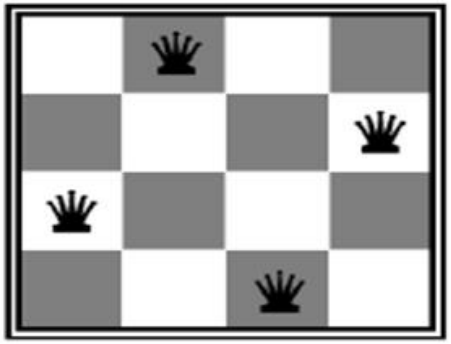
$$(WA, NT) \in \{(red, green), (red, blue), (green, red), \ldots\}$$

- Solutions are assignments satisfying all constraints, e.g.:

$$\{WA = red, NT = green, Q = red, \\ NSW = green, V = red, SA = blue, T = green\}$$

Example: N-Queens

- Formulation 1:
  - Variables: X_{ij}
  - Domains: $\{0, 1\}$
  - Constraints

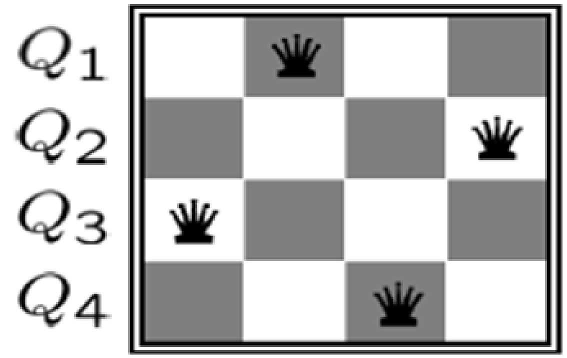


$$\forall i, j, k \ \ (X_{ij}, X_{ik}) \in \{(0,0), (0,1), (1,0)\}$$
$$\forall i, j, k \ \ (X_{ij}, X_{kj}) \in \{(0,0), (0,1), (1,0)\}$$
$$\forall i, j, k \ \ (X_{ij}, X_{i+k,j+k}) \in \{(0,0), (0,1), (1,0)\}$$
$$\forall i, j, k \ \ (X_{ij}, X_{i+k,j-k}) \in \{(0,0), (0,1), (1,0)\}$$

$$\sum_{i,j} X_{ij} = N$$

Example: N-Queens

- Formulation 2:
  - Variables: Q_k
  - Domains: $\{1, 2, 3, \ldots N\}$
  - Constraints:



Implicit: $\forall i, j$ non-threatening(Q_i, Q_j)

-or-

Explicit: $(Q_1, Q_2) \in \{(1,3), (1,4), \ldots\}$

$\cdots$

Constraint Graph

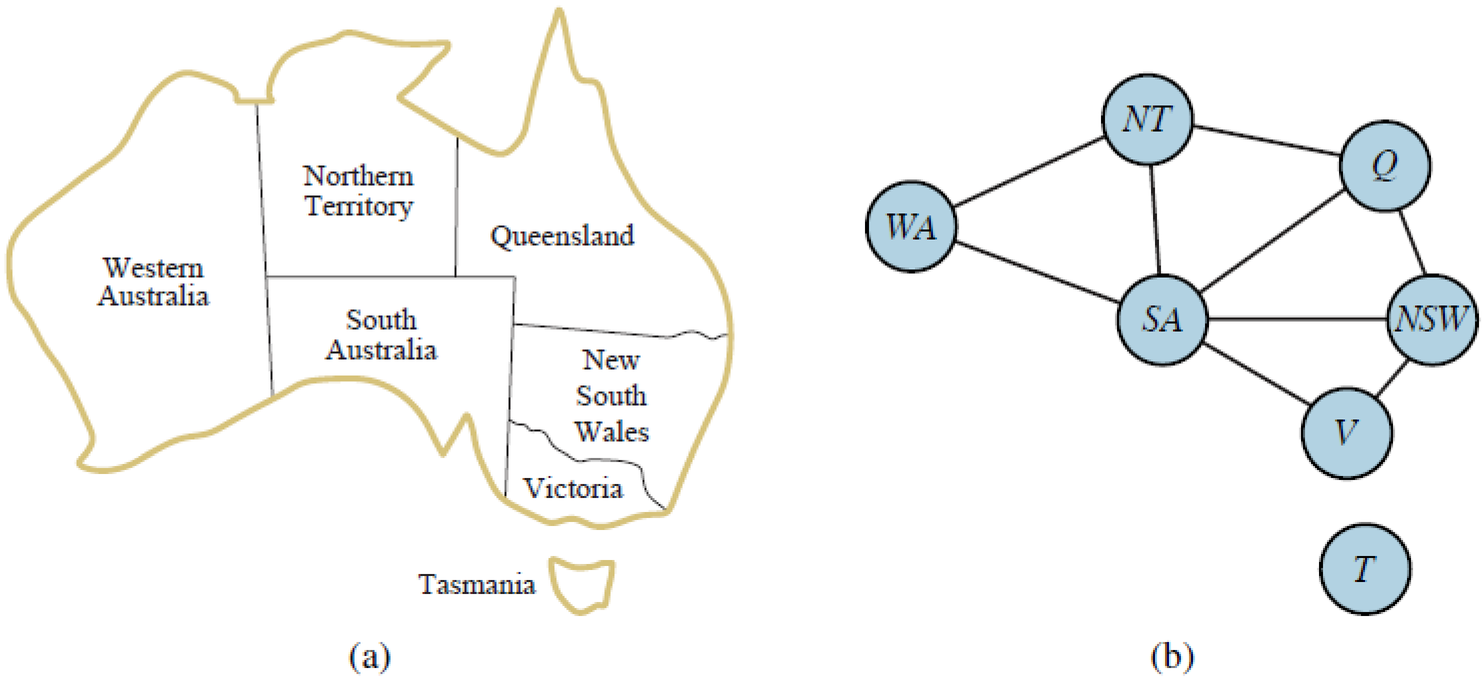


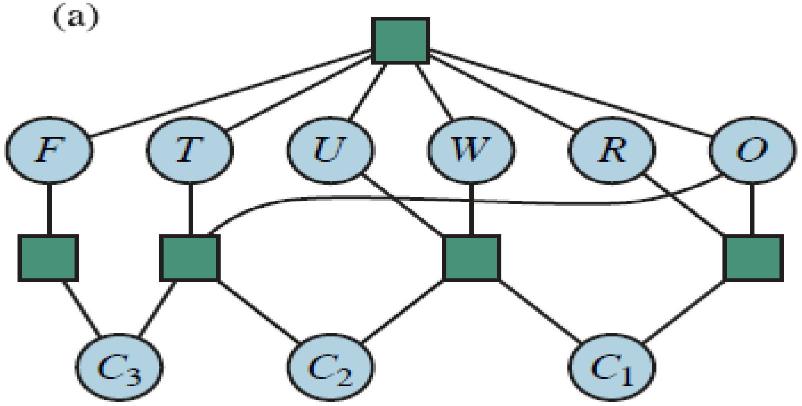
Figure 6.1 (a) The principal states and territories of Australia. Coloring this map can be viewed as a constraint satisfaction problem (CSP). The goal is to assign colors to each region so that no neighboring regions have the same color. (b) The map-coloring problem represented as a constraint graph.

Example CSP: Cryptarithmetic

- Variables (circles):
  $F\ T\ U\ W\ R\ O\ X_1\ X_2\ X_3$
- Domains:
  $\{0,1,2,3,4,5,6,7,8,9\}$
- Constraints (boxes):
  $\text{alldiff}(F,T,U,W,R,O)$
  $O+O=R+10\cdot X_1$
  $\cdots$

$$\begin{array}{cccc} & T & W & O \\ + & T & W & O \\ \hline F & O & U & R \end{array}$$

(a)



(b)

Figure 6.2 (a) A cryptarithmetic problem. Each letter stands for a distinct digit; the aim is to find a substitution of digits for letters such that the resulting sum is arithmetically correct, with the added restriction that no leading zeroes are allowed. (b) The constraint hypergraph for the cryptarithmetic problem, showing the *Alldiff* constraint (square box at the top) as well as the column addition constraints (four square boxes in the middle). The variables C_1, C_2, and C_3 represent the carry digits for the three columns from right to left.

Example: Sudoku

	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

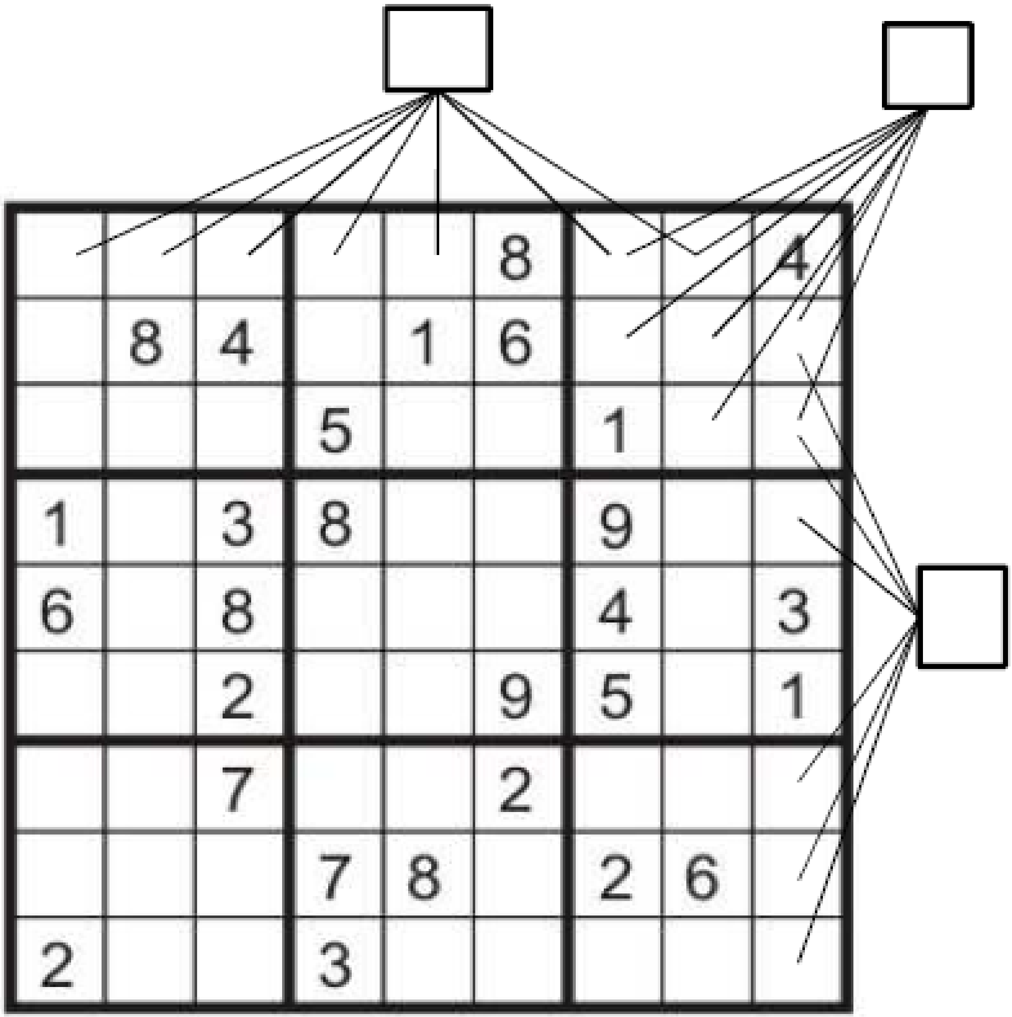
(a)

	1	2	3	4	5	6	7	8	9
A	4	8	3	9	2	1	6	5	7
B	9	6	7	3	4	5	8	2	1
C	2	5	1	8	7	6	4	9	3
D	5	4	8	1	3	2	9	7	6
E	7	2	9	5	6	4	1	3	8
F	1	3	6	7	9	8	2	4	5
G	3	7	2	6	8	9	5	1	4
H	8	1	4	2	5	3	7	6	9
I	6	9	5	4	1	7	3	8	2

(b)

Figure 6.4 (a) A Sudoku puzzle and (b) its solution.

Example: Sudoku



- Variables:
  - Each (open) square
- Domains:
  - {1,2,…,9}
- Constraints:

9-way alldiff for each column

9-way alldiff for each row

9-way alldiff for each region

(or can have a bunch of pairwise inequality constraints)

Varieties of CSPs

- Discrete Variables
  - Finite domains
    - Size d means $O(d^n)$ complete assignments
    - E.g., Boolean CSPs, including Boolean satisfiability (NP-complete)
  - Infinite domains (integers, strings, etc.)
    - E.g., job scheduling, variables are start/end times for each job
    - Linear constraints solvable, nonlinear undecidable

- Continuous variables
  - E.g., start/end times for Hubble Telescope observations
  - Linear constraints solvable in polynomial time by LP methods

Varieties of Constraints

- Varieties of Constraints
  - Unary constraints involve a single variable (equiv. to shrinking domains):

$$SA \neq green$$

  - Binary constraints involve pairs of variables:

$$SA \neq WA$$

  - Higher-order constraints involve 3 or more variables:
    e.g., cryptarithmetic column constraints

- Preferences (soft constraints):
  - E.g., red is better than green
  - Often representable by a cost for each variable assignment
  - Gives constrained optimization problems
  - (We'll ignore these until we get to Bayes' nets)

Real World CSPs

- Assignment problems: e.g., who teaches what class
- Timetabling problems: e.g., which class is offered when and where?
- Hardware configuration
- Transportation scheduling
- Factory scheduling
- Floorplanning
- Fault diagnosis
- … lots more!

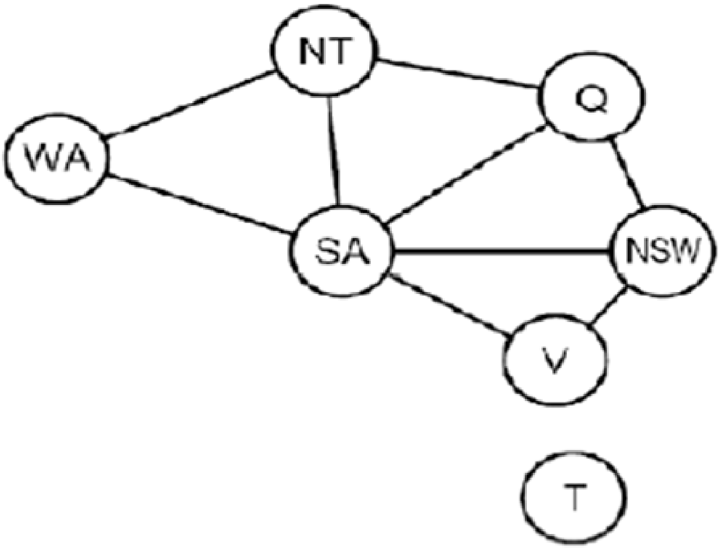
- Many real-world problems involve real-valued variables…

Standard Search Formulation

- Standard search formulation of CSPs (incremental)
- Let's start with the straightforward, dumb 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
  - Goal test: the current assignment is complete and satisfies all constraints
- Simplest CSP ever: two bits, constrained to be equal

Search Methods

- What would BFS do?



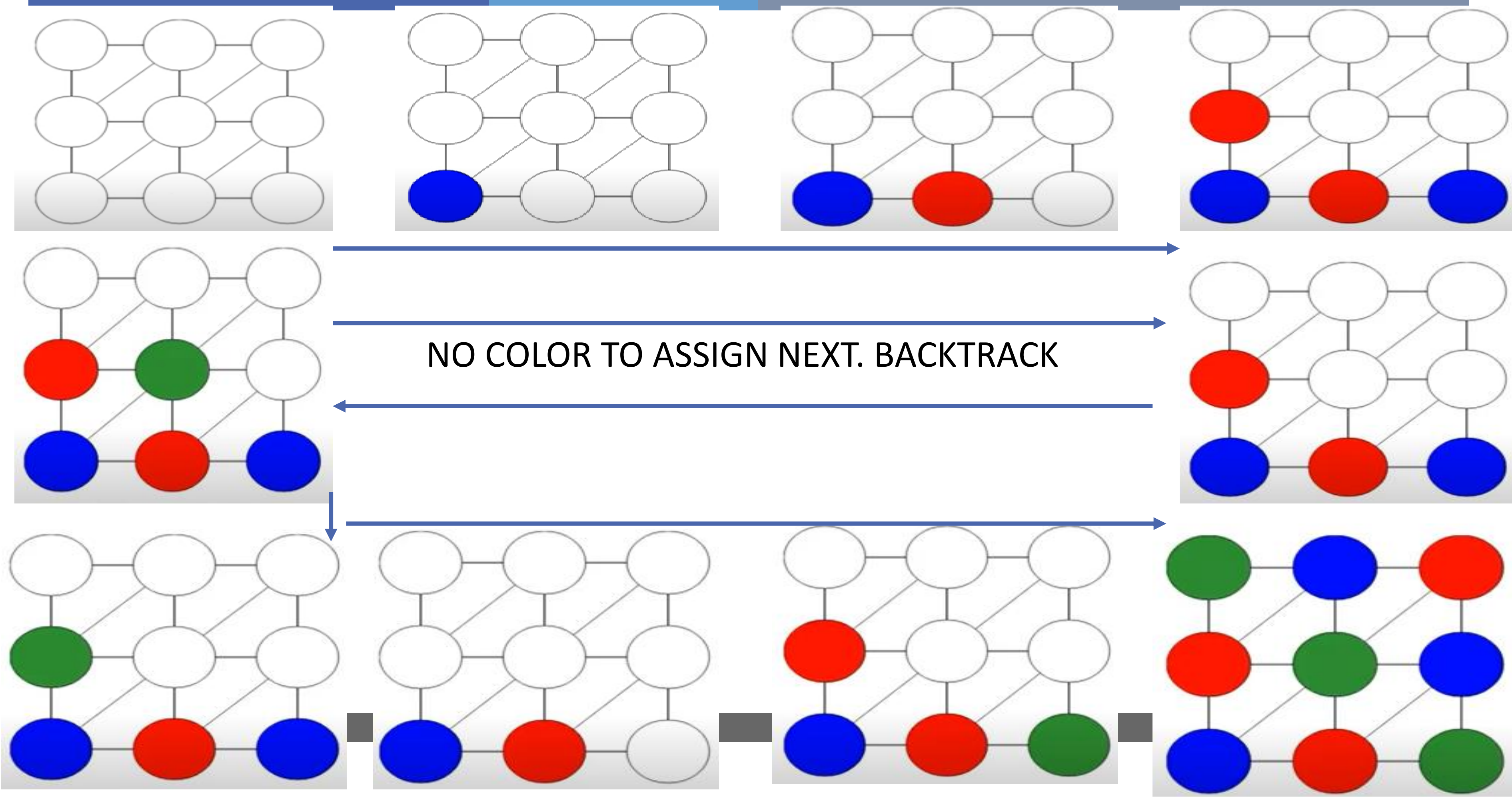
- What would DFS do?
- What problems does this approach have?

Backtracking Search

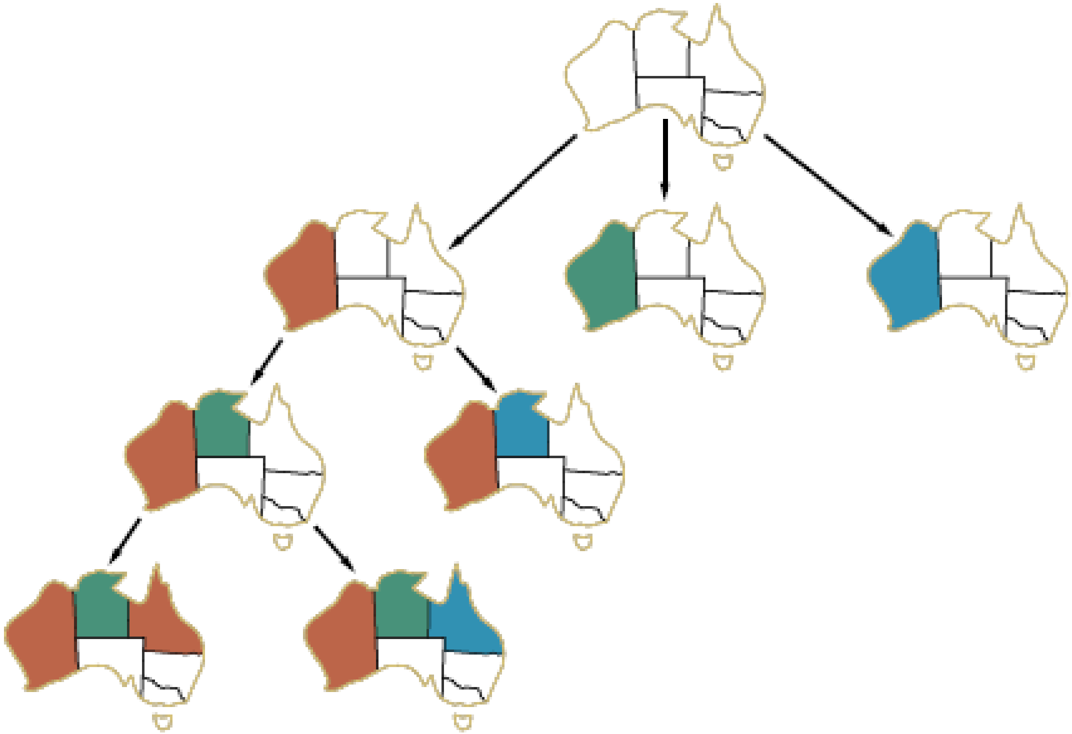
- Idea 1: Only consider a single variable at each point
  - Variable assignments are commutative, so fix ordering
  - I.e., [WA = red then NT = green] same as [NT = green then WA = red]
  - Only need to consider assignments to a single variable at each step
  - How many leaves are there?
- Idea 2: Only allow legal assignments at each point
  - I.e. consider only values which do not conflict previous assignments
  - Might have to do some computation to figure out whether a value is ok
  - "Incremental goal test"
- Depth-first search for CSPs with these two improvements is called *backtracking search* (useless name, really)
  - [DEMO]
- Backtracking search is the basic uninformed algorithm for CSPs
- Can solve n-queens for $n \approx 25$

Demonstration

Assigning Value from Domain = { Blue, Red, Green} To Each Variables (circles) from Left to Right Order



Backtracking Example



Part of the search tree for the map-coloring problem in Figure

Backtracking Search

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

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

Improving Backtracking

- General-purpose ideas give huge gains in speed
- Ordering:
  - Which variable should be assigned next?
  - In what order should its values be tried?
- Filtering: Can we detect inevitable failure early?
- Structure: Can we exploit the problem structure?

Filtering: Forward Checking

- Idea: Keep track of remaining legal values for unassigned variables (using immediate constraints)
- Idea: Terminate when any variable has no legal values

Progress of Map Coloring

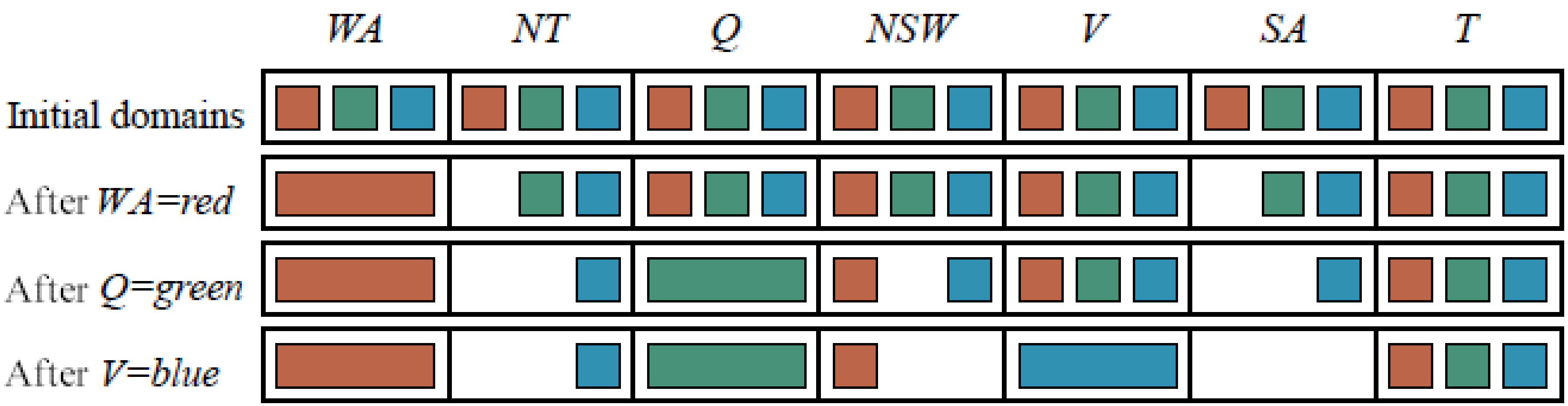
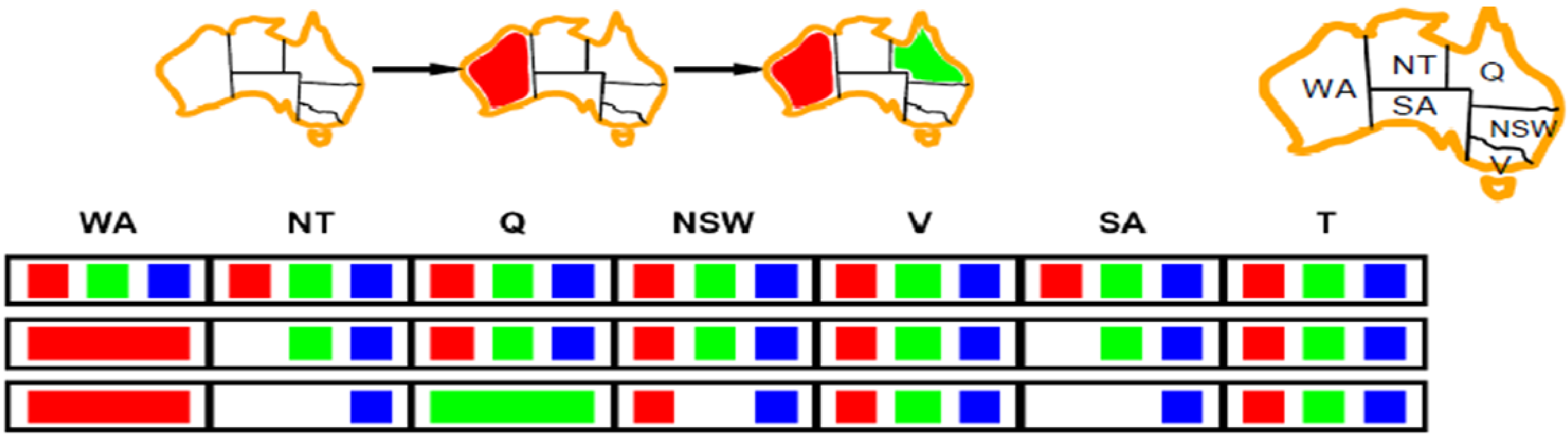


Figure 6.7 The progress of a map-coloring search with forward checking. $WA = red$ is assigned first; then forward checking deletes *red* from the domains of the neighboring variables NT and SA. After $Q = green$ is assigned, *green* is deleted from the domains of NT, SA, and NSW. After $V = blue$ is assigned, *blue* is deleted from the domains of NSW and SA, leaving SA with no legal values.

Filtering: Constraint Propagation

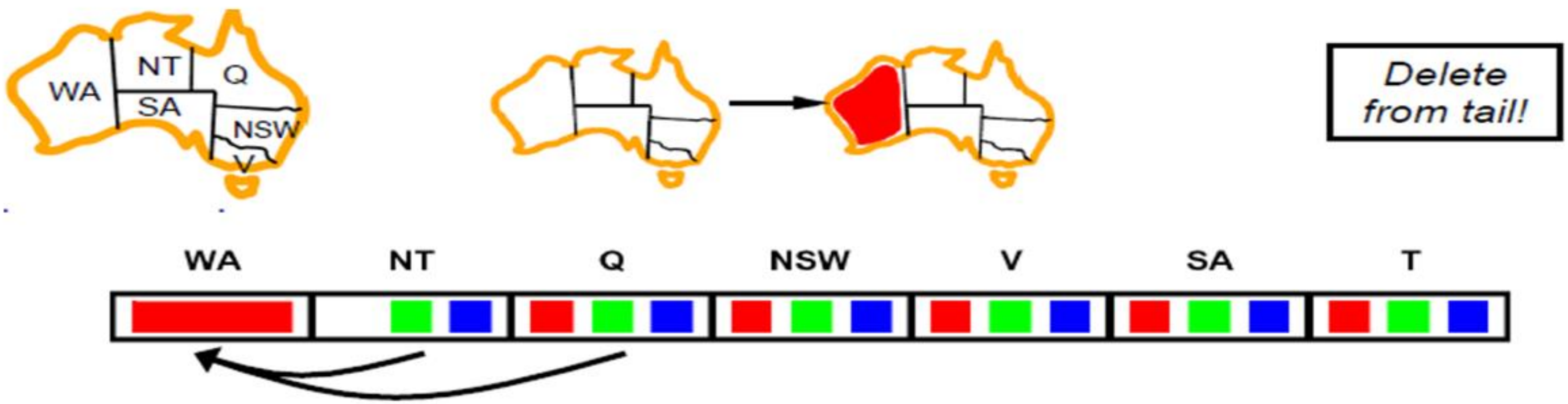
- Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures:



- NT and SA cannot both be blue!
- Why didn't we detect this yet?
- *Constraint propagation* propagates from constraint to constraint

Consistency of An Arc

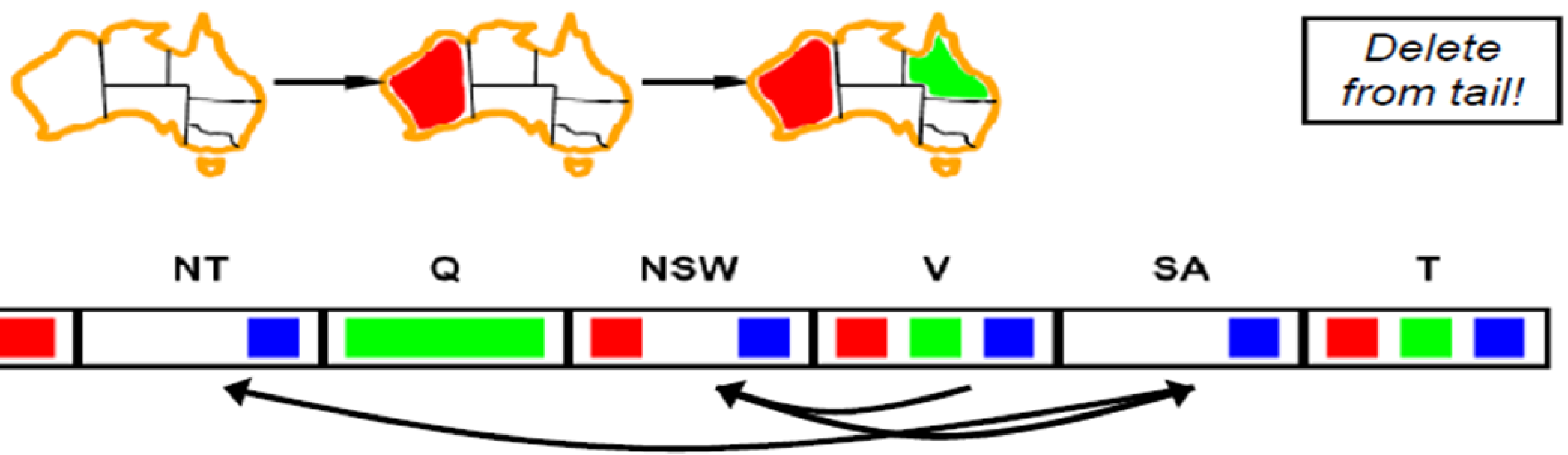
- An arc X → Y is consistent iff for *every* x in the tail there is *some* y in the head which could be assigned without violating a constraint



- What happens?
- Forward checking = Enforcing consistency of each arc pointing to the new assignment

Arc Consistency of a CSP

- A simple form of propagation makes sure all arcs are consistent:



- If X loses a value, neighbors of X need to be rechecked!
- Arc consistency detects failure earlier than forward checking
- What's the downside of enforcing arc consistency?
- Can be run as a preprocessor or after each assignment

Arc Consistency of a CSP

Variable	**Constraints**	**Constraints**
WA = {R, G, B}	SA≠WA	WA≠SA
NT= {R, G, B}	SA≠NT	NT≠SA
Q= {R, G, B}	SA≠Q	Q≠SA
NSW= {R, G, B}	SA≠NSW	NSW≠SA
V = {R, G, B}	SA≠V	V≠SA
SA= {R, G, B}	WA≠NT	NT≠WA
T= {R, G, B}	NT≠Q	Q≠NT
	Q≠NSW	NSW≠Q
	NSW≠V	V≠NSW

Arc Consistency of a CSP

Variable	Constraints	Constraints
WA = {R}	SA≠WA	WA≠SA
NT= {R, G, B}	SA≠NT	NT≠SA
Q= {R, G, B}	SA≠Q	Q≠SA
NSW= {R, G, B}	SA≠NSW	NSW≠SA
V = {R, G, B}	SA≠V	V≠SA
SA= {R, G, B}	WA≠NT	NT≠WA
T= {R, G, B}	NT≠Q	Q≠NT
	Q≠NSW	NSW≠Q
	NSW≠V	V≠NSW

Arc Consistency of a CSP

Variable	Constraints	Constraints
WA = {R}	SA≠WA	WA≠SA
NT= {R, G, B}	SA≠NT	NT≠SA
Q= {R, G, B}	SA≠Q	Q≠SA
NSW= {R, G, B}	SA≠NSW	NSW≠SA
V = {R, G, B}	SA≠V	V≠SA
SA= {R, G, B}	WA≠NT	NT≠WA
T= {R, G, B}	NT≠Q	Q≠NT
	Q≠NSW	NSW≠Q
	NSW≠V	V≠NSW

Arc Consistency of a CSP

Variable	**Constraints**	**Constraints**
WA = {R}	SA≠WA	WA≠SA
NT= {R, G, B}	SA≠NT	NT≠SA
Q= {R, G, B}	SA≠Q	Q≠SA
NSW= {R, G, B}	SA≠NSW	NSW≠SA
V = {R, G, B}	SA≠V	V≠SA
SA= {G, B}	WA≠NT	NT≠WA
T= {R, G, B}	NT≠Q	Q≠NT
	Q≠NSW	NSW≠Q
	NSW≠V	V≠NSW

Arc Consistency of a CSP

Variable	**Constraints**	**Constraints**
WA = {R}	SA≠WA	WA≠SA
NT= {R, G, B}	SA≠NT	NT≠SA
Q= {R, G, B}	SA≠Q	Q≠SA
NSW= {R, G, B}	SA≠NSW	NSW≠SA
V = {R, G, B}	SA≠V	V≠SA
SA= {G, B}	WA≠NT	NT≠WA
T= {R, G, B}	NT≠Q	Q≠NT
	Q≠NSW	NSW≠Q
	NSW≠V	V≠NSW

Arc Consistency of a CSP

Variable	Constraints		Constraints
WA = {R}	SA≠WA		WA≠SA
NT= {R, G, B}	SA≠NT		NT≠SA
Q= {R, G, B}	SA≠Q		Q≠SA
NSW= {R, G, B}	SA≠NSW		NSW≠SA
V = {R, G, B}	SA≠V		V≠SA
SA= {G, B}	WA≠NT	➡	NT≠WA
T= {R, G, B}	NT≠Q		Q≠NT
	Q≠NSW		NSW≠Q
	NSW≠V		V≠NSW

Arc Consistency of a CSP

Variable	Constraints	Constraints
WA = {R}	SA≠WA	WA≠SA
NT= {G, B}	SA≠NT	NT≠SA
Q= {R, G, B}	SA≠Q	Q≠SA
NSW= {R, G, B}	SA≠NSW	NSW≠SA
V = {R, G, B}	SA≠V	V≠SA
SA= {G, B}	WA≠NT	NT≠WA
T= {R, G, B}	NT≠Q	Q≠NT
	Q≠NSW	NSW≠Q
	NSW≠V	V≠NSW

Arc Consistency of a CSP

Variable	Constraints		Constraints
WA = {R}	SA≠WA		WA≠SA
NT= {G, B}	SA≠NT		NT≠SA
Q= {R, G, B}	SA≠Q		Q≠SA
NSW= {R, G, B}	SA≠NSW		NSW≠SA
V = {R, G, B}	SA≠V		V≠SA
SA= {G, B}	WA≠NT		NT≠WA
T= {R, G, B}	NT≠Q	➡	Q≠NT
	Q≠NSW	➡	NSW≠Q
	NSW≠V	➡	V≠NSW

Arc Consistency of a CSP

Variable	Constraints	Constraints
WA = {R}	SA≠WA	WA≠SA
NT= {G, B}	SA≠NT	NT≠SA
Q= {R, G, B}	SA≠Q	Q≠SA
NSW= {R, G, B}	SA≠NSW	NSW≠SA
V = {R, G, B}	SA≠V	V≠SA
SA= {G, B}	WA≠NT	NT≠WA
T= {R, G, B}	NT≠Q	Q≠NT
	Q≠NSW	NSW≠Q
	NSW≠V	V≠NSW

Arc Consistency of a CSP

Variable	Constraints	Constraints
WA = {R}	SA≠WA	WA≠SA
NT= {G, B}	SA≠NT	NT≠SA
Q= {R, G, B}	SA≠Q	Q≠SA
NSW= {R, G, B}	SA≠NSW	NSW≠SA
V = {R, G, B}	SA≠V	V≠SA
SA= {G, B}	WA≠NT	NT≠WA
T= {R, G, B}	NT≠Q	Q≠NT
	Q≠NSW	NSW≠Q
	NSW≠V	V≠NSW

Arc Consistency of a CSP

Variable	Constraints	Constraints
WA = {R}	SA≠WA	WA≠SA
NT= {G, B}	SA≠NT	NT≠SA
Q= {G}	SA≠Q	Q≠SA
NSW= {R, G, B}	SA≠NSW	NSW≠SA
V = {R, G, B}	SA≠V	V≠SA
SA= {G, B}	WA≠NT	NT≠WA
T= {R, G, B}	NT≠Q	Q≠NT
	Q≠NSW	NSW≠Q
	NSW≠V	V≠NSW

Arc Consistency of a CSP

Variable	Constraints	Constraints
WA = {R}	SA≠WA	WA≠SA
NT= {G, B}	SA≠NT	NT≠SA
Q= {G}	SA≠Q	Q≠SA
NSW= {R, G, B}	SA≠NSW	NSW≠SA
V = {R, G, B}	SA≠V	V≠SA
SA= {B}	WA≠NT	NT≠WA
T= {R, G, B}	NT≠Q	Q≠NT
	Q≠NSW	NSW≠Q
	NSW≠V	V≠NSW

Arc Consistency of a CSP

Variable	**Constraints**	**Constraints**
WA = {R}	SA≠WA	WA≠SA
NT= {B}	SA≠NT	NT≠SA
Q= {G}	SA≠Q	Q≠SA
NSW= {R, G, B}	SA≠NSW	NSW≠SA
V = {R, G, B}	SA≠V	V≠SA
SA= {B}	WA≠NT	NT≠WA
T= {R, G, B}	NT≠Q	Q≠NT
	Q≠NSW	NSW≠Q
	NSW≠V	V≠NSW

Arc Consistency of a CSP

Variable	Constraints	Constraints
WA = {R}	SA≠WA	WA≠SA
NT= {B}	SA≠NT	NT≠SA
Q= {G}	SA≠Q	Q≠SA
NSW= {R, G, B}	SA≠NSW	NSW≠SA
V = {R, G, B}	SA≠V	V≠SA
SA= {B}	WA≠NT	NT≠WA
T= {R, G, B}	NT≠Q	Q≠NT
	Q≠NSW	NSW≠Q
	NSW≠V	V≠NSW

Arc Consistency of a CSP

Variable	**Constraints**	**Constraints**
WA = {R}	SA≠WA	WA≠SA
NT= {B}	SA≠NT	NT≠SA
Q= {G}	SA≠Q	Q≠SA
NSW= {R, B}	SA≠NSW	NSW≠SA
V = {R, G, B}	SA≠V	V≠SA
SA= {B}	WA≠NT	NT≠WA
T= {R, G, B}	NT≠Q	Q≠NT
	Q≠NSW	NSW≠Q
	NSW≠V	V≠NSW

Arc Consistency of a CSP

Variable	Constraints	Constraints
WA = {R}	SA≠WA	WA≠SA
NT= {B}	SA≠NT	NT≠SA
Q= {G}	SA≠Q	Q≠SA
NSW= {R, B}	SA≠NSW	NSW≠SA
V = {R, G, B}	SA≠V	V≠SA
SA= {B}	WA≠NT	NT≠WA
T= {R, G, B}	NT≠Q	Q≠NT
	Q≠NSW	NSW≠Q
	NSW≠V	V≠NSW

Arc Consistency of a CSP

Variable	Constraints	Constraints
WA = {R}	SA≠WA	WA≠SA
NT= {B}	SA≠NT	NT≠SA
Q= {G}	SA≠Q	Q≠SA
NSW= {R, B}	SA≠NSW	NSW≠SA
V = {R, G, B}	SA≠V	V≠SA
SA= {B}	WA≠NT	NT≠WA
T= {R, G, B}	NT≠Q	Q≠NT
	Q≠NSW	NSW≠Q
	NSW≠V	V≠NSW

Arc Consistency of a CSP

Variable	**Constraints**	**Constraints**
WA = {R}	SA≠WA	WA≠SA
NT= {B}	SA≠NT	NT≠SA
Q= {G}	SA≠Q	Q≠SA
NSW= {R, B}	SA≠NSW	NSW≠SA
V = {R, G, B}	SA≠V	V≠SA
SA= {}	WA≠NT	NT≠WA
T= {R, G, B}	NT≠Q	Q≠NT
	Q≠NSW	NSW≠Q
	NSW≠V	V≠NSW

Arc Consistency

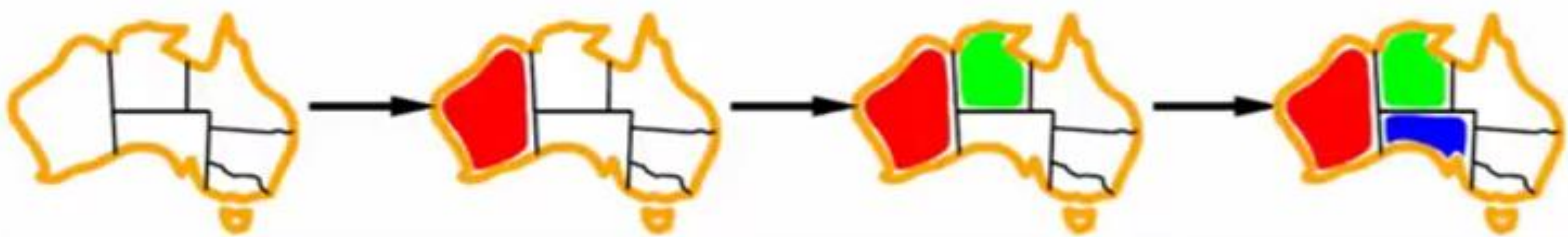
```
function AC-3( csp) returns the CSP, possibly with reduced domains
    inputs: csp, a binary CSP with variables {X1, X2, ..., Xn}
    local variables: queue, a queue of arcs, initially all the arcs in csp

    while queue is not empty do
        (Xi, Xj) ← REMOVE-FIRST(queue)
        if REMOVE-INCONSISTENT-VALUES(Xi, Xj) then
            for each Xk in NEIGHBORS[Xi] do
                add (Xk, Xi) to queue
```

```
function REMOVE-INCONSISTENT-VALUES( Xi, Xj) returns true iff succeeds
    removed ← false
    for each x in DOMAIN[Xi] do
        if no value y in DOMAIN[Xj] allows (x,y) to satisfy the constraint Xi ↔ Xj
            then delete x from DOMAIN[Xi];  removed ← true
    return removed
```

- Runtime: $O(n^2d^3)$, can be reduced to $O(n^2d^2)$
- … but detecting all possible future problems is NP-hard – why?

Ordering: Minimum Remaining Value

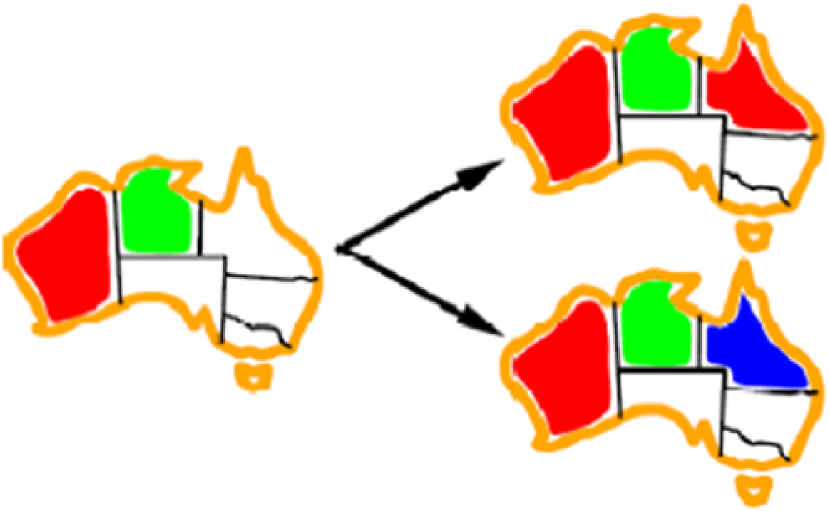
- Minimum remaining values (MRV):
  - Choose the variable with the fewest legal values



- Why min rather than max?
- Also called "most constrained variable"
- "Fail-fast" ordering

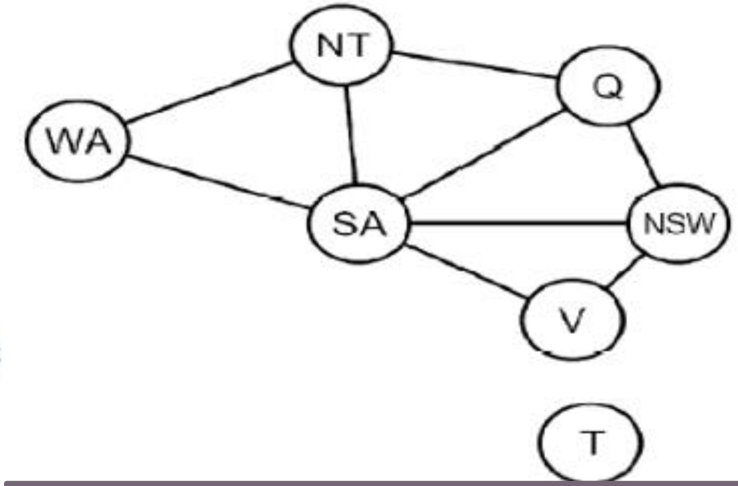
Ordering:
Least Constraining Value

- Given a choice of variable:
  - Choose the *least constraining value*
  - The one that rules out the fewest values in the remaining variables
  - Note that it may take some computation to determine this!
- Why least rather than most?
- Combining these heuristics makes 1000 queens feasible



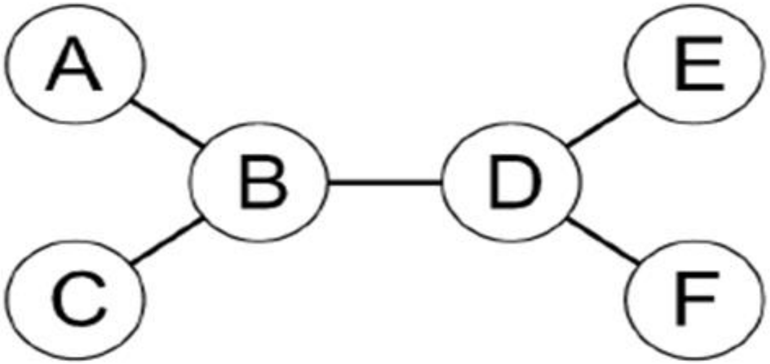
Problem Structure

- Tasmania and mainland are independent subproblems
- Identifiable as connected components of constraint graph
- Suppose each subproblem has c variables out of n total
  - Worst-case solution cost is $O((n/c)(d^c))$, linear in n
  - E.g., n = 80, d = 2, c =20
  - 2^{80} = 4 billion years at 10 million nodes/sec
  - $(4)(2^{20})$ = 0.4 seconds at 10 million nodes/sec



Dividing a Boolean CSP with 80 variables into four sub problems reduces the worst-case solution time from the lifetime of the universe down to less than a second

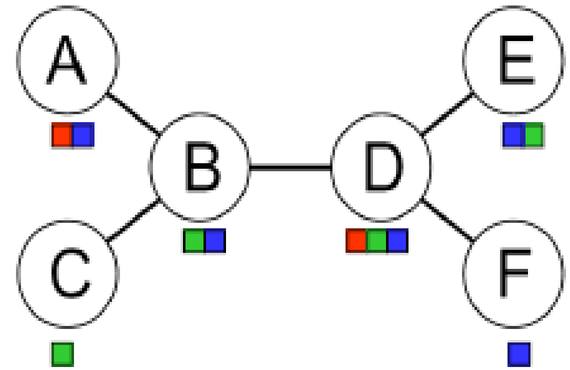
Tree Structure CSPs

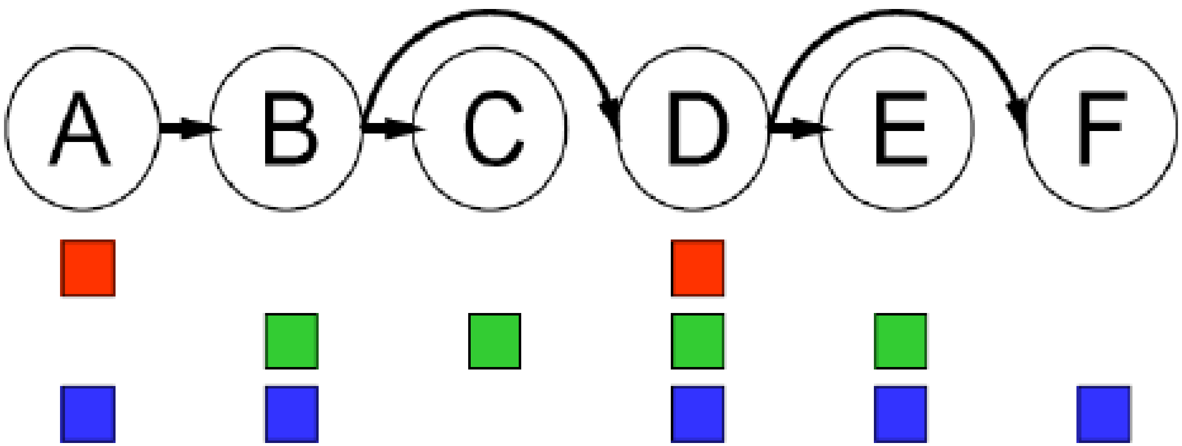


- Theorem: if the constraint graph has no loops, the CSP can be solved in $O(n\ d^2)$ time
  - Compare to general CSPs, where worst-case time is $O(d^n)$
- This property also applies to probabilistic reasoning (later): an important example of the relation between syntactic restrictions and the complexity of reasoning.

Tree-Structured CSPs

- Choose a variable as root, order variables from root to leaves such that every node's parent precedes it in the ordering

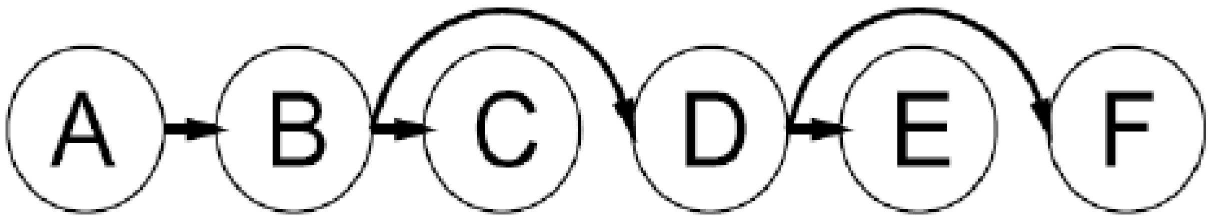




- For i = n : 2, apply RemoveInconsistent(Parent(X_i),X_i)
- For i = 1 : n, assign X_i consistently with Parent(X_i)
- Runtime: $O(n\ d^2)$ (why?)

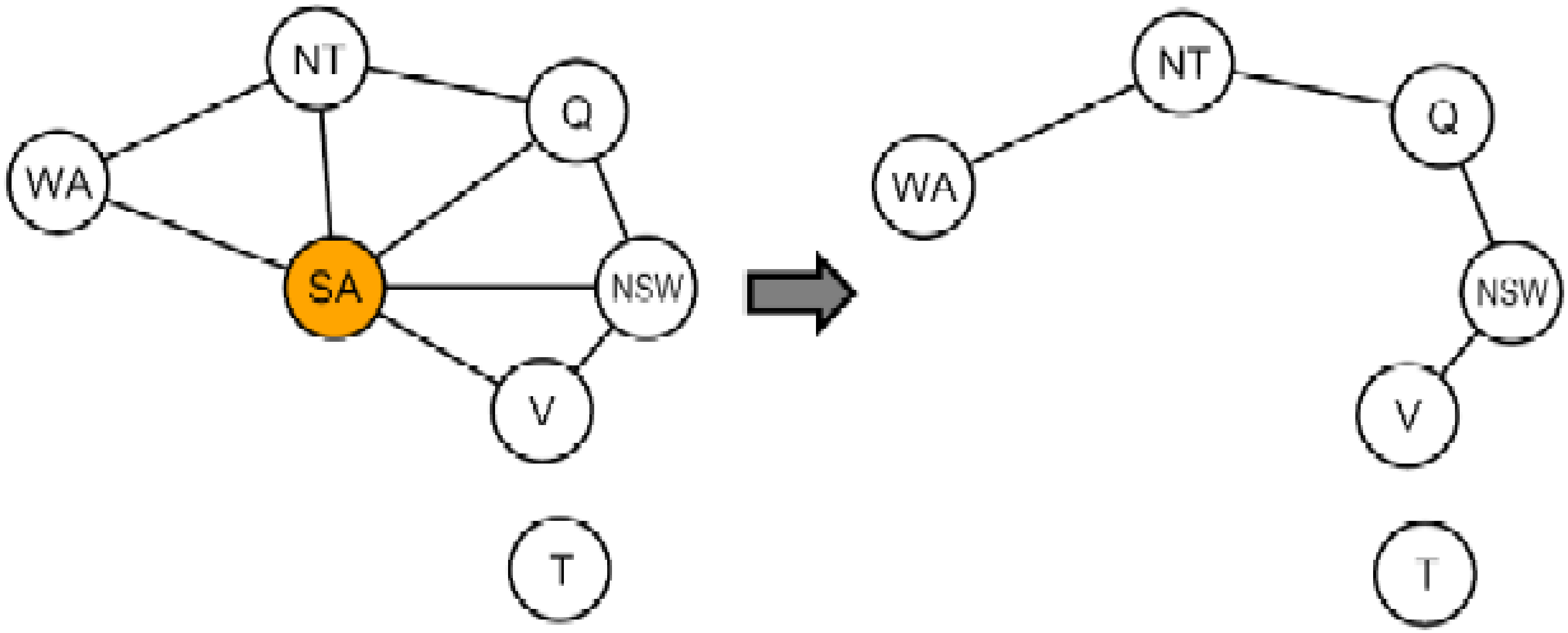
Tree-Structured CSPs

- Why does this work?
- Claim: After processing the right k nodes, given any satisfying assignment to the rest, the right k can be assigned (left to right) without backtracking
- Proof: Induction on position



- Why doesn't this algorithm work with loops?
- Note: we'll see this basic idea again with Bayes' nets

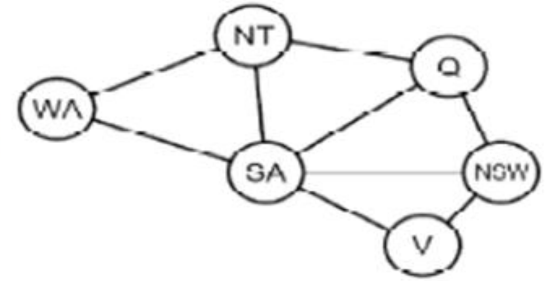
Nearly Tree-Structured CSPs

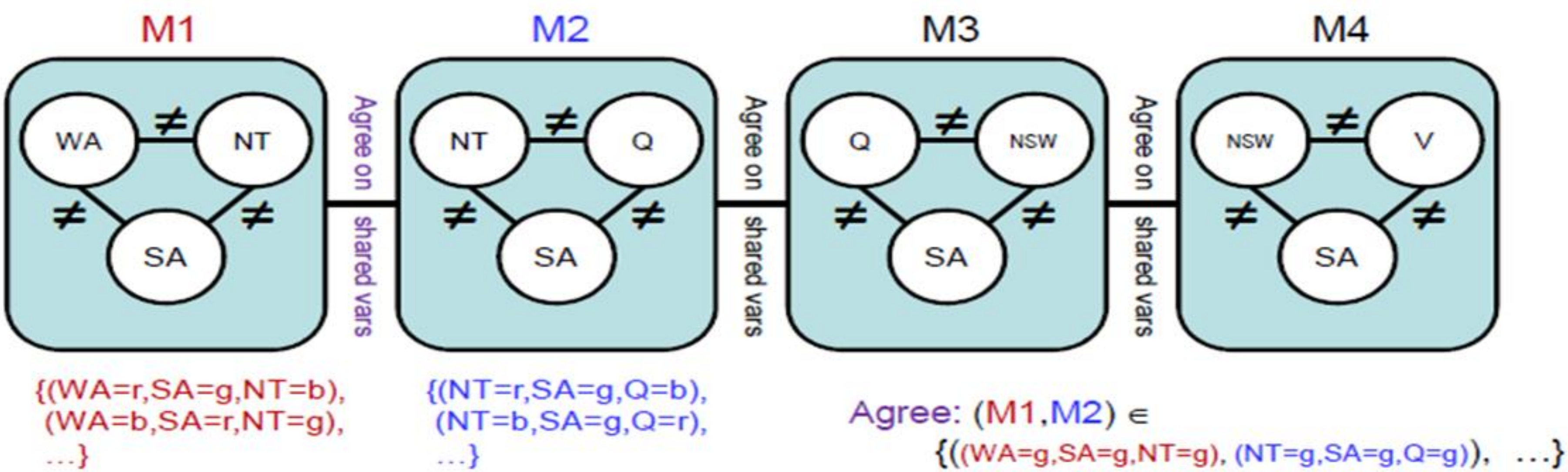


- Conditioning: instantiate a variable, prune its neighbors' domains
- Cutset conditioning: instantiate (in all ways) a set of variables such that the remaining constraint graph is a tree
- Cutset size c gives runtime O((d^c) (n-c) d^2), very fast for small c

Tree Decomposition

- Create a tree-structured graph of overlapping subproblems, each is a mega-variable
- Solve each subproblem to enforce local constraints
- Solve the CSP over subproblem mega-variables using our efficient tree-structured CSP algorithm





Summary

- CSPs are a special kind of search problem:
  - States defined by values of a fixed set of variables
  - Goal test defined by constraints on variable values
- Backtracking = depth-first search with one legal variable assigned per node
- Variable ordering and value selection heuristics help significantly
- Forward checking prevents assignments that guarantee later failure
- Constraint propagation (e.g., enforcing arc consistency) does additional work to constrain values and detect inconsistencies
- Constraint graphs allow for analysis of problem structure
- Tree-structured CSPs can be solved in linear time
- Iterative min-conflicts is usually effective in practice

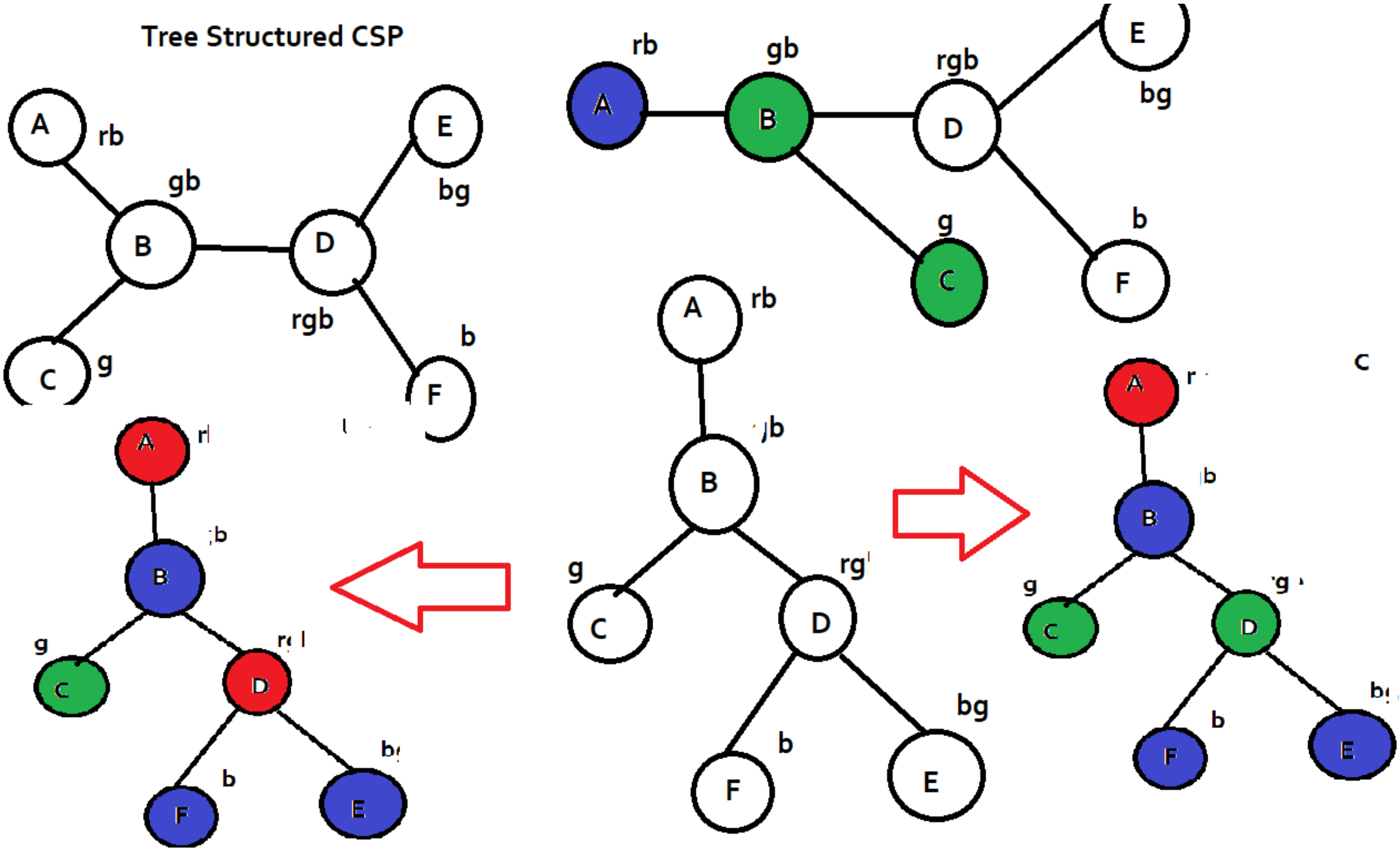
References

1. Chapter 6: Introduction , Pages 202-220
"Artificial Intelligence: A Modern Approach," by Stuart J. Russell and Peter Norvig,

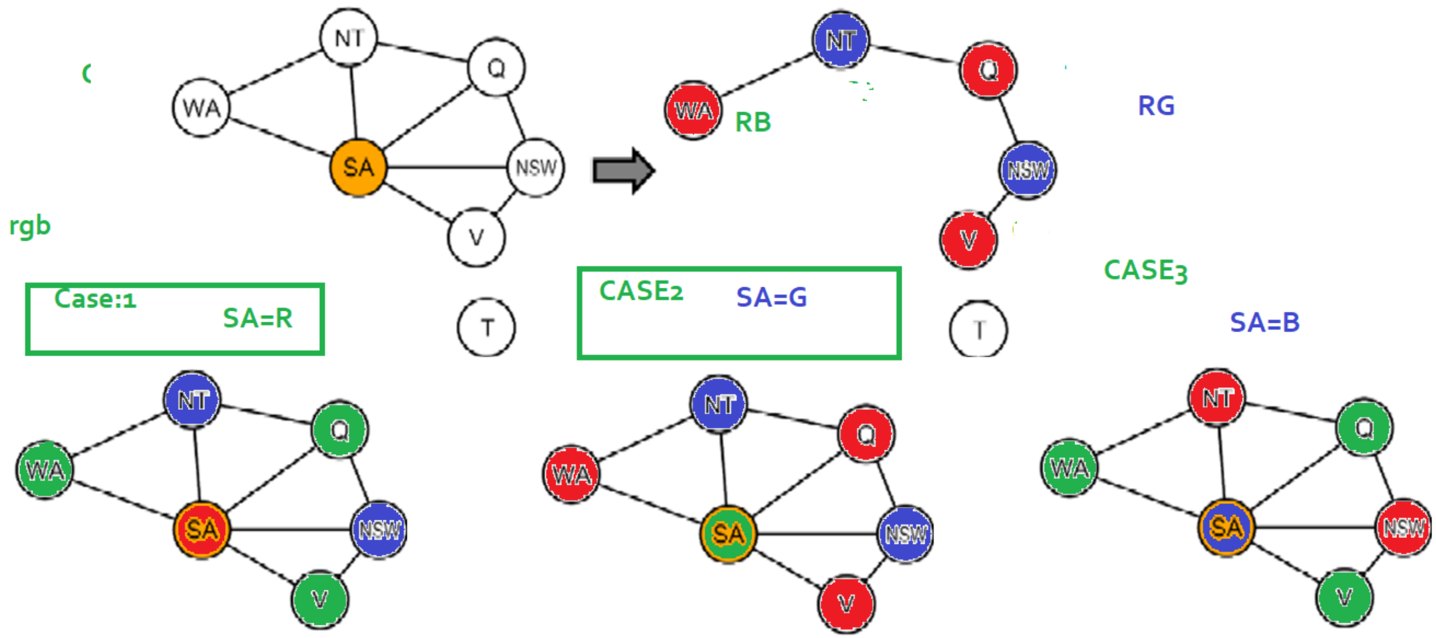
Books

1. “Artificial Intelligence: A Modern Approach,” by Stuart J. Russell and Peter Norvig.
2. "Artificial Intelligence: Structures and Strategies for Complex Problem Solving", by George F. Luger, (2002)
3. "Artificial Intelligence: Theory and Practice", by Thomas Dean.
4. "AI: A New Synthesis", by Nils J. Nilsson.
5. “Programming for machine learning,” by J. Ross Quinlan,
6. “Neural Computing Theory and Practice,” by Philip D. Wasserman, .
7. “Neural Network Design,” by Martin T. Hagan, Howard B. Demuth, Mark H. Beale, .
8. “Practical Genetic Algorithms,” by Randy L. Haupt and Sue Ellen Haupt.
9. “Genetic Algorithms in Search, optimization and Machine learning,” by David E. Goldberg.
10. "Computational Intelligence: A Logical Approach", by David Poole, Alan Mackworth, and Randy Goebel.
11. “Introduction to Turbo Prolog”, by Carl Townsend.

Tree Structured CSP



Nearly Tree-Structured CSPs



D={~~0~~,~~1~~,2,3,4,~~5~~,~~6~~,7,8,~~9~~}

9

	c4	c3	c2	c1
	S	E	N	D
+	M	O	R	E
M	O	N	E	Y

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

S	9
E	5
N	6
D	7
M	1
O	0
R	8
Y	2