Artificial Intelligence

Lecture 4： Constraint Satisfaction Problems

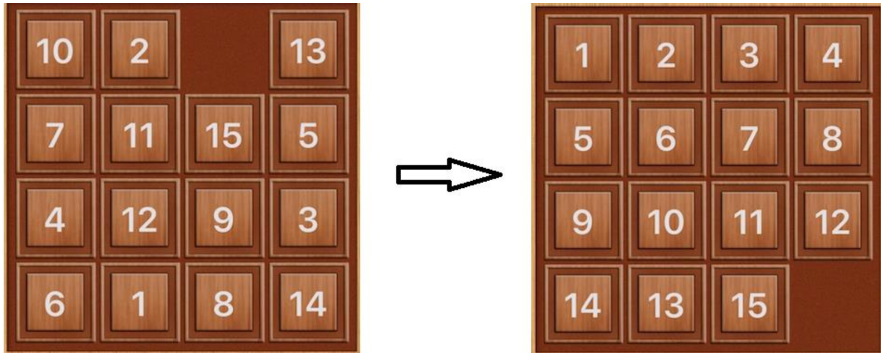
Xiaojin Gong

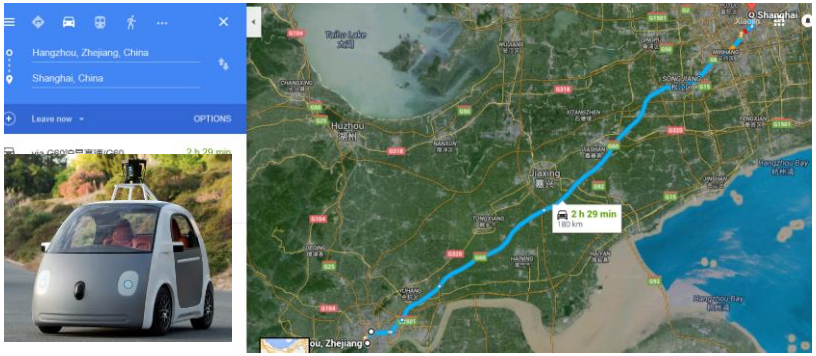
2022-03-14

Credits: AI Courses in Berkeley

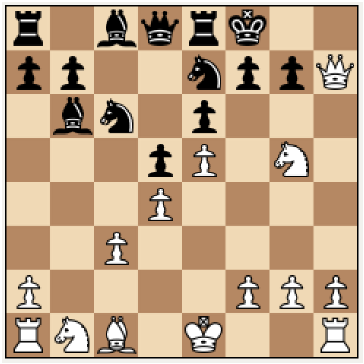
Review

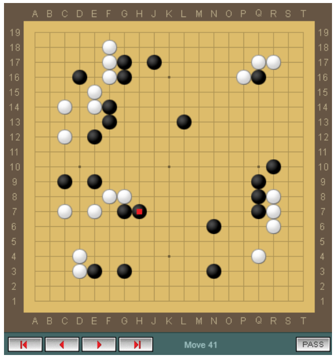
- Search problems





- Adversarial search





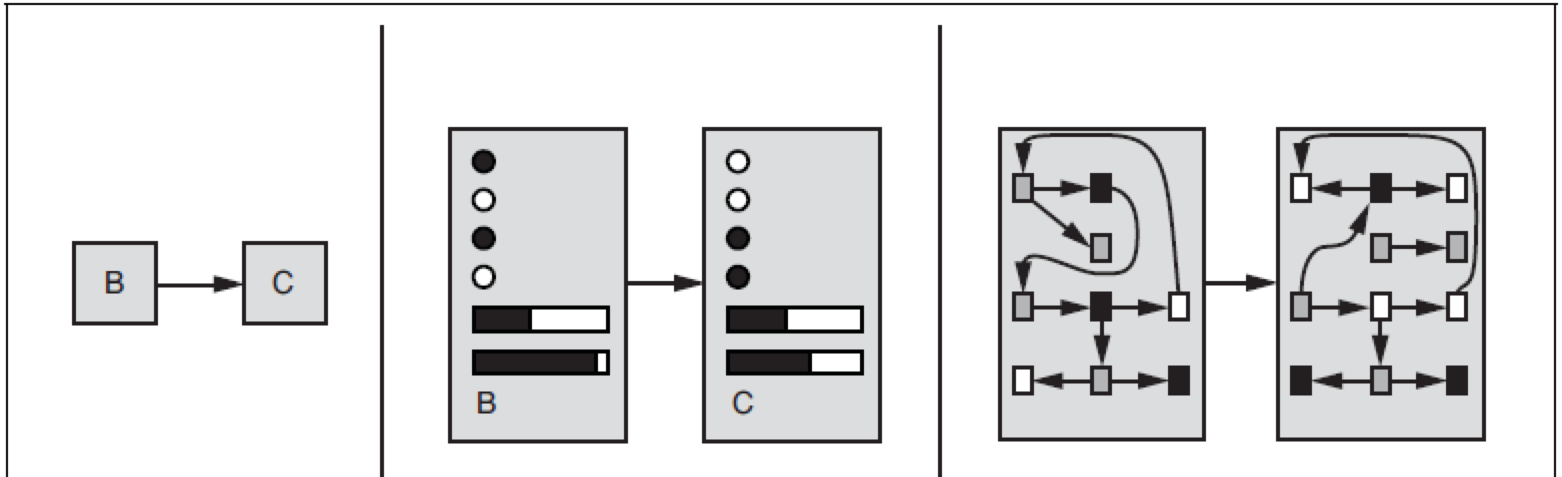
Review

- Rational Agents
  - Structure
  - Representation

Search
Markov decision processes

Constraint satisfaction
Bayesian network

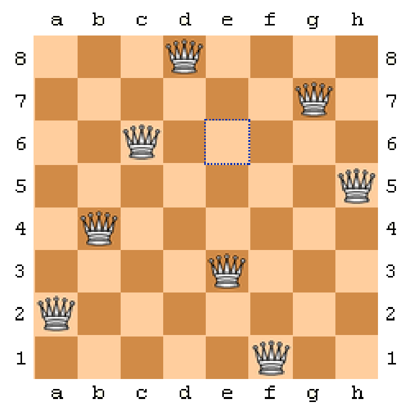
Knowledge-based learning
Natural language processing



(a) Atomic (b) Factored (b) Structured

Outline

- Constraint satisfaction problems
  - Problem formulation
  - Backtracking search
  - Inference in CSPs



	5	3	2		7			8
6		1	5					2
2			9	1	3		5	
7	1	4	6	9	2			
	2						6	
			4	5	1	2	9	7
	6		3	2	5			9
1					6	3		4
8			1		9	6	7	

- A special subset of search problems
- The goal itself is important, not the path

Constraint Satisfaction Problem

- A CPS consists of three components:
  - A set of variables $X = \{X_1, \quad \cdots, \quad X_n\}$
  - A set of domains $D = \{D_1, \quad \cdots, \quad D_n\}$
  - A set of constraints C
- State is defined by an assignment of values from a domain $D_i$to some or all of the variables X_i.
  - 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.
- Goal test is a set of constraints specifying allowable combinations of values for subsets of variables.
- Solution is a consistent, complete assignment.

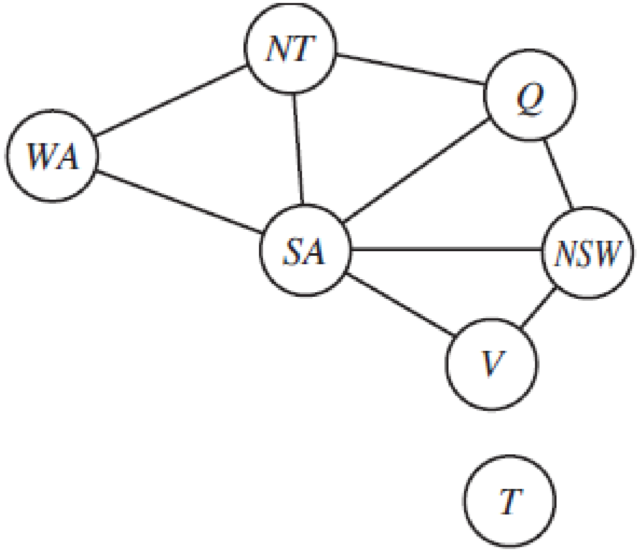
Example: Map Coloring

- Variables: $X = \{WA, NT, SA, Q, NSW, V, T\}$
- Domains: $D = \{r, g, b\}$
- Constraints:
  - Implicit: $WA \neq NT, \cdots$
  - Explicit: $(WA, NT) \in \{(r, g), (r, b), \cdots\}, \ldots$

- Solutions are assignments satisfying all constraints

Constraint Graph

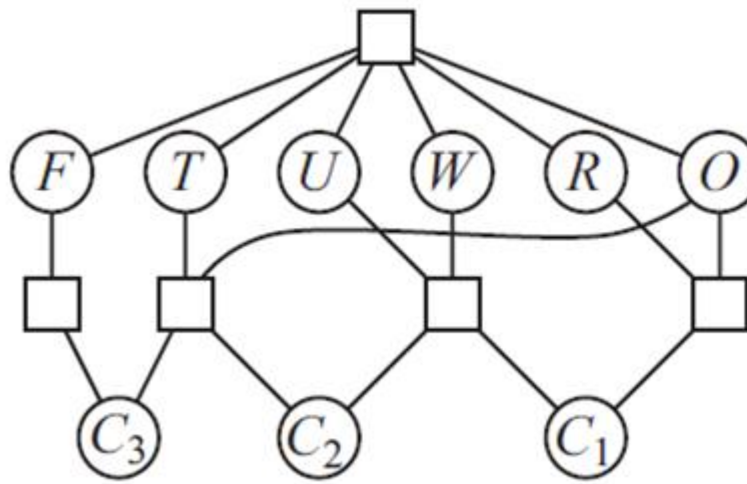
- Each node corresponds to a variable
- Each arc participates in a constraint
- General-purpose CSP algorithms use the graph structure to speed up search.



Example: Cryptarithmetic Puzzle

- Variables: $X = \{F, T, U, W, R, O, C_1, C_2, C_3\}$
- Domains: $D = \{0,1,2,\cdots,9\}$
- Constraints:
  - $Alldiff(F, T, U, W, R, O)$
  - $O + O = R + 10C_1$
  - $C_1 + W + W = U + 10C_2$
  - $C_2 + T + T = O + 10C_3$
  - $C_3 = F$

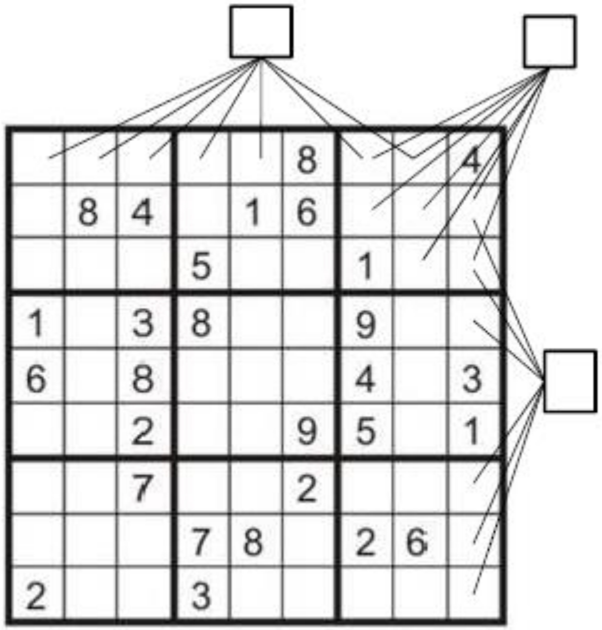
$$\begin{array}{cccc} & T & W & O \\ + & T & W & O \\ \hline F & O & U & R \end{array}$$



Constraint Hypergraph

Example: Sudoku

- Variables: Each (open) square
- Domains: $D = \{1, 2, \cdots, 9\}$
- Constraints:
  - 9-way *Alldiff* for each column
  - 9-way *Alldiff* for each row
  - 9-way *Alldiff* for each region



- Solutions are assignments satisfying all constraints

Example: Job-shop Scheduling

- Car assembly 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
- Variables:

$$\begin{aligned} X = \{ & Axle_F, Axle_B, Wheel_{RF}, Wheel_{LF}, Wheel_{RB}, Wheel_{LB}, Nuts_{RF}, \\ & Nuts_{LF}, Nuts_{RB}, Nuts_{LB}, Cap_{RF}, Cap_{LF}, Cap_{RB}, Cap_{LB}, Inspect\} . \end{aligned}$$

- Domains: $D = \{1, 2, \cdots, 27\}$
- Constraints:
  - Precedence constraint

$$Axle_F + 10 \leq Wheel_{RF}; \quad Wheel_{RF} + 1 \leq Nuts_{RF}; \quad Nuts_{RF} + 2 \leq Cap_{RF};$$

  - Disjunctive constraint

$$(Axle_F + 10 \leq Axle_B) \quad \textbf{or} \quad (Axle_B + 10 \leq Axle_F)$$

Example: Real World CSPs

- Timetabling problems
- Transportation scheduling
- Factory scheduling
- Hardware configuration
- …

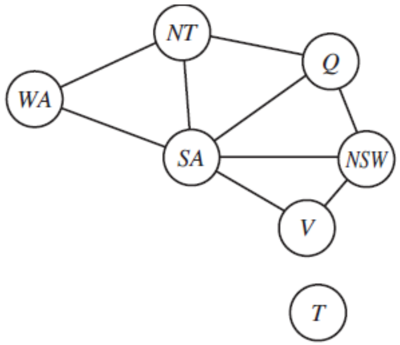
Many real-world problems involve real-valued variables

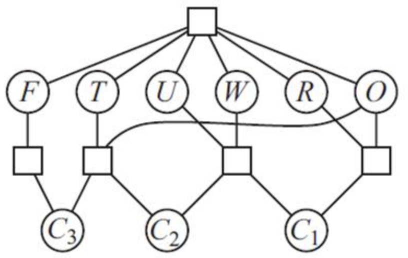
Varieties of CSPs: Variables

- Discrete variables
  - Finite domains
    - E.g., map coloring, Sudoku
  - Infinite domains
    - E.g., job scheduling
    - Linear constraints solvable, nonlinear undecidable
- Continuous variables
  - E.g., start/end times for Hubble Telescope observations
  - Linear constraints solvable in polynomial time by linear programming methods

Varieties of CSPs: Constraints

- Varieties of constraints
  - Unary constraints
    - E.g.,map coloring, $SA \neq g$
  - Binary constraints
    - E.g., map coloring, $SA \neq WA$
  - Higher-order constraints
    - E.g., cryptarithmetic column constraints





- Preference constraints (Soft constraints)
  - Indicating which solutions are preferred
  - E.g., red is better than green
  - Often represented by a cost for each variable assignment
  - Gives constrained optimization problems

Solving CSPs

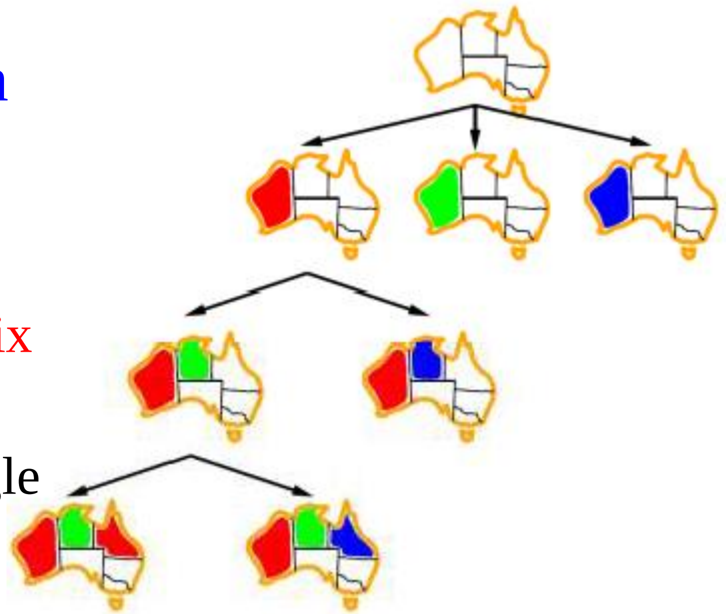
- Standard search formulation
  - States defined by the values assigned so far
  - Initial state: the empty assignment
  - Successor function: assign a value to an unassigned variable
  - Goal test: if the assignment is complete and satisfies all constraints
- Search tree
  - For *n* variables of domain size *d*,
    - Top level: nd
    - 2nd level: $(n-1)d$
    - Leaves: $n!d^n$

Commutativity in CPS!

- Depth-first search

Backtracking Search for CSPs

- Backtracking search is the basic uninformed algorithm for solving CSPs
- Backtracking search is the depth-first search with two improvements:
  1. One variable at a time
     - Variable assignments are commutative, so fix ordering
     - Only need to consider assignments to a single variable at each step
  2. Backtrack when a variable has no legal values
     - Incremental goal test
- The number of leaves in the search tree is d^n



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
```

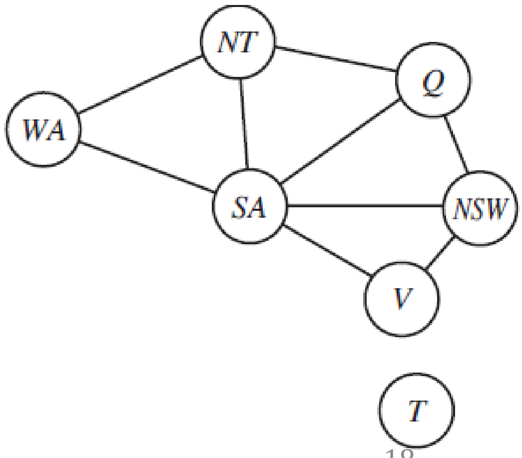
Backtracking Search for CSPs

- Ordering:
  - Which variable should be assigned next?
  - In what order its values be tried?
- Inference:
  - Infer reductions in the domain of variables
- Structure:
  - Can we exploit the problem structure?

Variable and Value Ordering

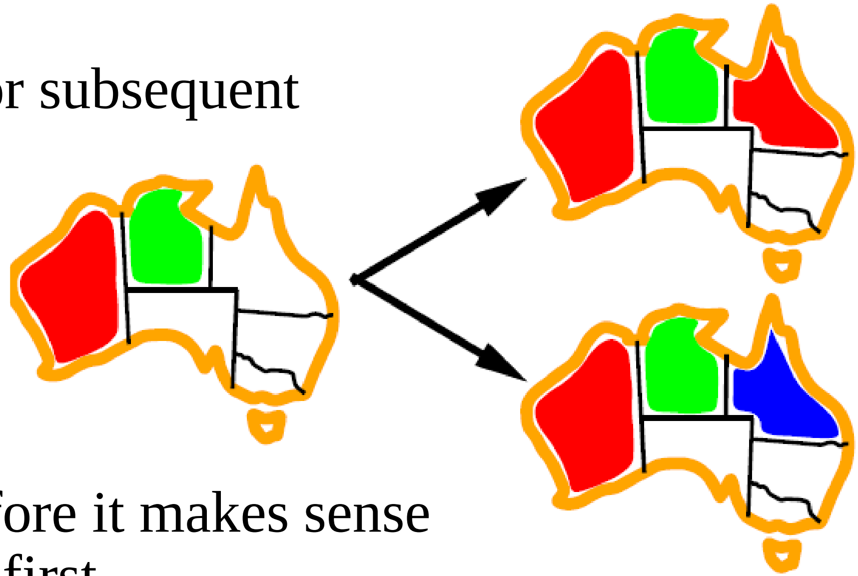
- Variable ordering:
  - Minimum-remaining-values (MRV) heuristic
    - Choose the variable with the fewest “legal” values
    - Also called the most constrained variable or fail-first heuristic
    - Help minimize the number of nodes in the search tree by pruning larger parts of the tree earlier.

  - Degree heuristic
    - Choose the variable with the largest degree

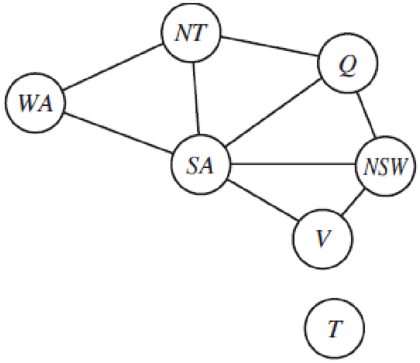


Variable and Value Ordering

- Value ordering:
  - Least-constraining-value heuristic
    - Leave the maximum flexibility for subsequent variable assignments
    - Fail-last heuristic



    - We only need one solution, therefore it makes sense to look for the most likely values first.



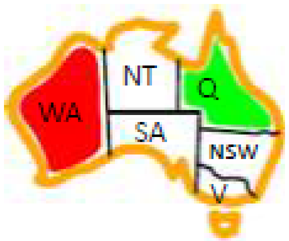
Inference in CSPs

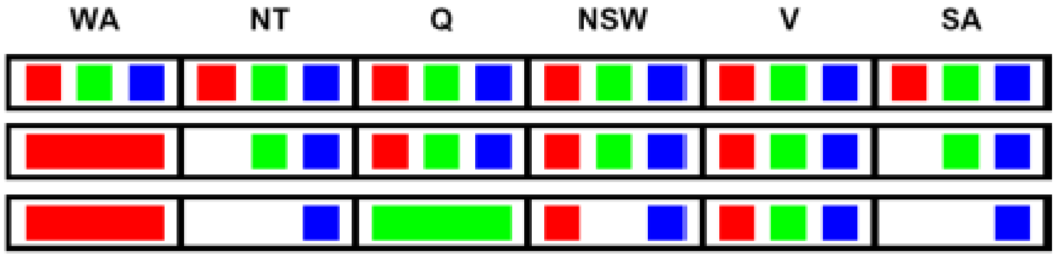
- Inference:
  - Constraint propagation
  - Reduce the number of legal values for a variable
  - Constraint propagation may be interwined with search, or done as a preprocessing step
  - The key idea is local consistency
- Forms of inference:
  - Forward checking
  - Maintaining Arc Consistency (MAC)

for each *value* **in** ORDER-DOMAIN-VALUES(*var*, *assignment*, *csp*) **do**
    if *value* is consistent with *assignment* **then**
        add $\{var = value\}$ to *assignment*
        $inferences \leftarrow$ INFERENCE(*csp*, *var*, *value*)

Inference: Forward Checking

- Whenever a variable *X* is assigned, the forward-checking process establishes arc consistency for it

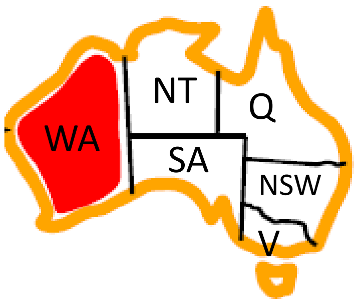


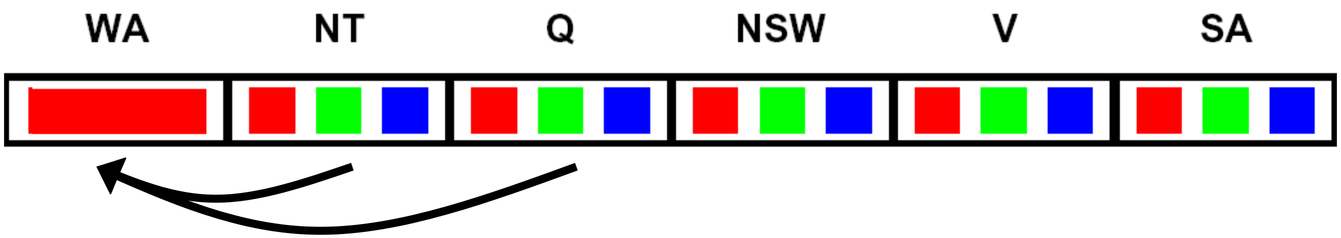


- Keep track of domains for unassigned variables and cross off bad options
- Forward checking propagates information from assigned to unassigned variables
- It makes the current variable arc-consistent, but doesn't look ahead and make all the other variables arc-consistent.

Arc Consistency

- An arc $X \rightarrow Y$ is consistent *iff* for every x there is some y which could be assigned without violating a constraint





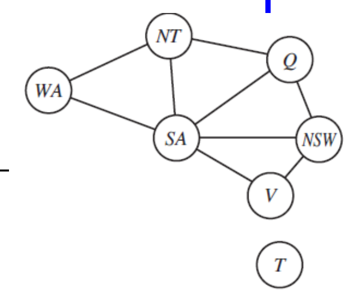
Delete from the tail!

- Forward checking: Enforcing consistency of arcs pointing to each new assignment

Arc-Consistency Algorithm (AC-3)

function AC-3(*csp*) **returns** false if an inconsistency is found and true otherwise
  inputs: *csp*, a binary CSP with components (X, D, C)
  local variables: *queue*, a queue of arcs, initially all the arcs in *csp*

  while *queue* is not empty **do**
    $(X_i, X_j) \leftarrow$ REMOVE-FIRST(*queue*)
    if REVISE(*csp*, X_i, X_j) **then**
      if size of $D_i = 0$ **then return** *false*
      for each X_k **in** X_i.NEIGHBORS - $\{X_j\}$ **do**
        add (X_k, X_i) to *queue*
  return *true*



---

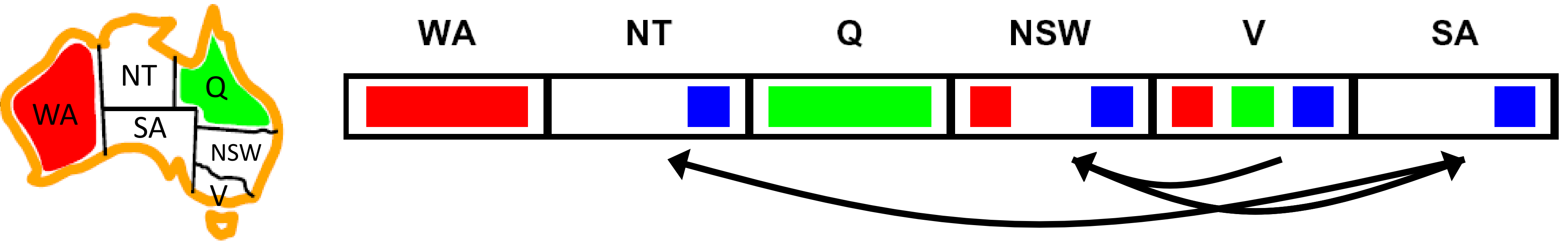
function REVISE(*csp*, X_i, X_j) **returns** true iff we revise the domain of X_i
  revised $\leftarrow$ *false*
  for each x **in** D_i **do**
    if no value y in D_j allows (x,y) to satisfy the constraint between X_i and X_j **then**
      delete x from D_i
      revised $\leftarrow$ *true*
  return *revised*

Arc-Consistency Algorithm (AC-3)

- Runtime:
  - Assume a CSP with n variables, d domain size, and c binary constraints
  - Each arc (X_k, X_i) can be inserted d times
  - Checking consistency of an arc takes $O(d^2)$ time
  - So the total worst-case time is $O(cd^3)$

Arc-Consistency Algorithm (AC-3)

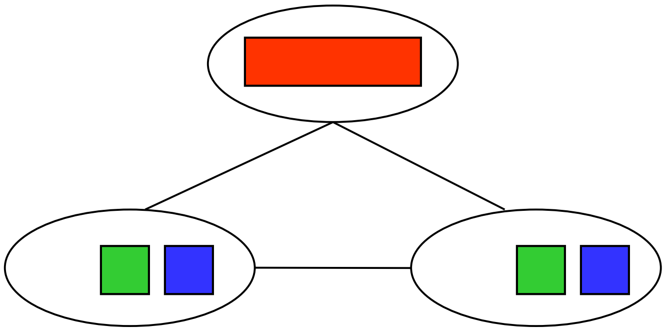
- A simple form of propagation makes sure all arcs are consistent:



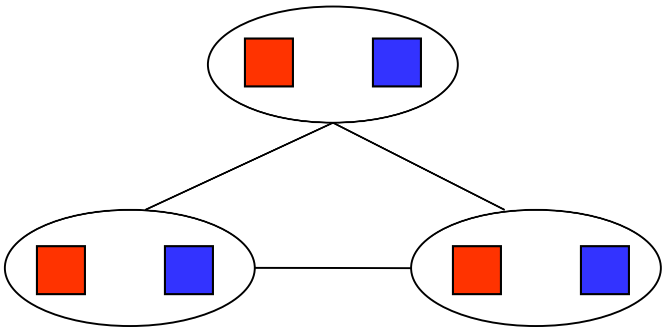
- Important: If X loses a value, neighbors of X need to be rechecked!
- Arc consistency detects failure earlier than forward checking
- Can be run as a preprocessor or after each assignment
- Interleaving inference with search

Limitations of Arc Consistency

- After enforcing arc consistency:
  - Can have one solution left
  - Can have multiple solutions left
  - Can have no solutions left (and not know it)



- Arc consistency still runs inside a backtracking search!



K-Consistency

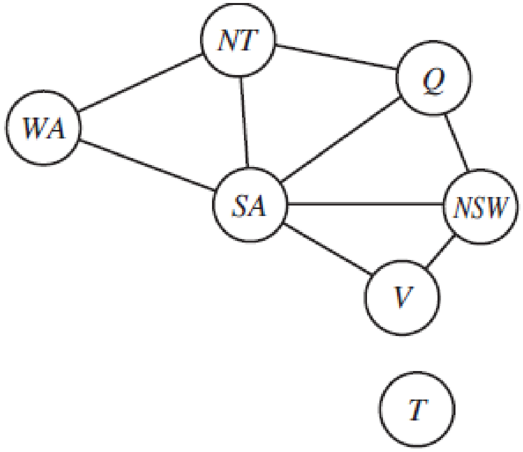
- Increasing degrees of consistency
  - 1-Consistency (Node Consistency):
    - Each single node's domain has a value which meets that node's unary constraints
  - 2-Consistency (Arc Consistency):
    - For each pair of nodes, any consistent assignment to one can be extended to the other
  - K-Consistency:
    - For each k nodes, any consistent assignment to k-1 can be extended to the kth node.
- Higher k more expensive to compute

Strong K-Consistency

- Strong k-consistency:
  - Also k-1, k-2, … 1 consistent
- Strong n-consistency means we can solve without backtrackin
  - Guaranteed to find a solution in $O(n^2d)$
    - Choose any assignment to any variable
    - Choose a new variable
    - By 2-consistency, there is a choice consistent with the first
    - Choose a new variable
    - By 3-consistency, there is a choice consistent with the first 2
    - …

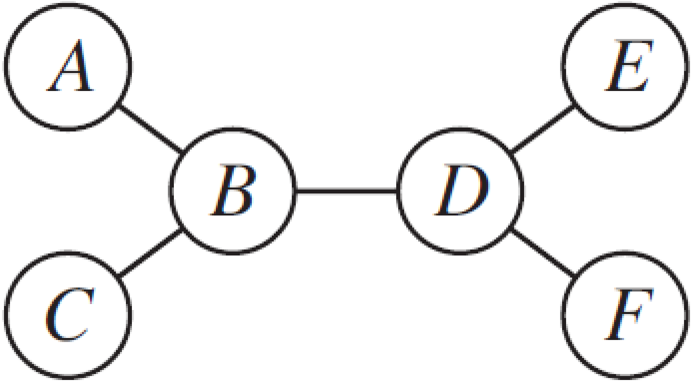
The Structure of Problems

- Decompose into independent subproblems
  - Independence can be ascertained by finding connected components
- Suppose each subproblem CSP_i has c variables from the total of n variables,
  - The total work is $O(n/c\ d^c)$
  - E.g. n=80, d=2, c=20
  - 2^{80}= 4 billion years at 10 million nodes/sec
  - $4 \cdot 2^{20}$ = 0.4 seconds at 10 million nodes/sec.



Tree-Structured CSP

- Any two variables are connected by only one path
- Any tree-structured CSP can be solved in $O(nd^2)$ time
  - Compare to general CSPs, where worst-case time is $O(d^n)$



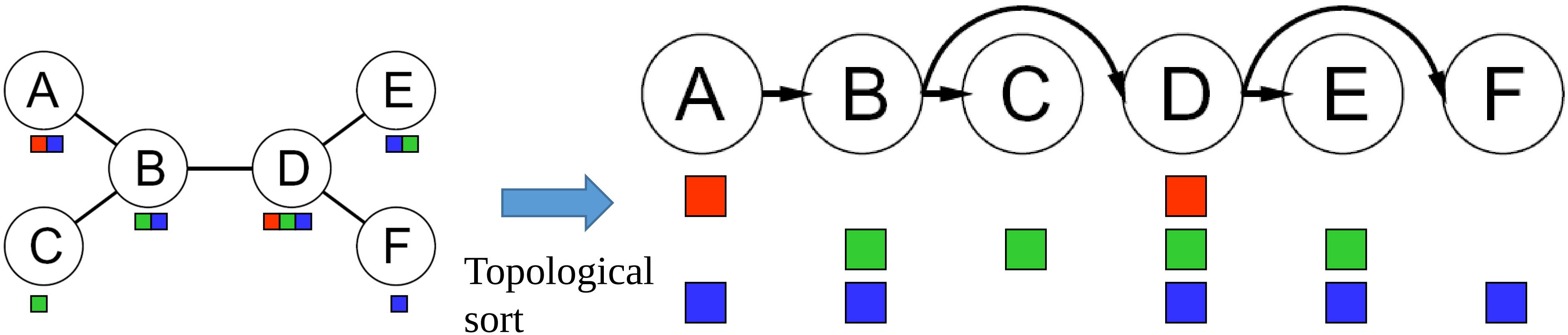
Tree-Structured CSP

```
function TREE-CSP-SOLVER( csp) returns a solution, or failure
   inputs: csp, a CSP with components X, D, C

   n ← number of variables in X
   assignment ← an empty assignment
   root ← any variable in X
   X ← TOPOLOGICALSORT(X, root)
   for j = n down to 2 do
     MAKE-ARC-CONSISTENT(PARENT(X_j), X_j)
     if it cannot be made consistent then return failure
   for i = 1 to n do
     assignment[X_i] ← any consistent value from D_i
     if there is no consistent value then return failure
   return assignment
```

Tree-Structured CSP

- Topological sort
  - Each variable appears after its parent in the tree.
- Directed arc consistency (DAC)
  - A CSP is DAC under an ordering $X_1, X_2, \ldots, X_n$ iff every X_i is arc-consistent with each X_j for j > I

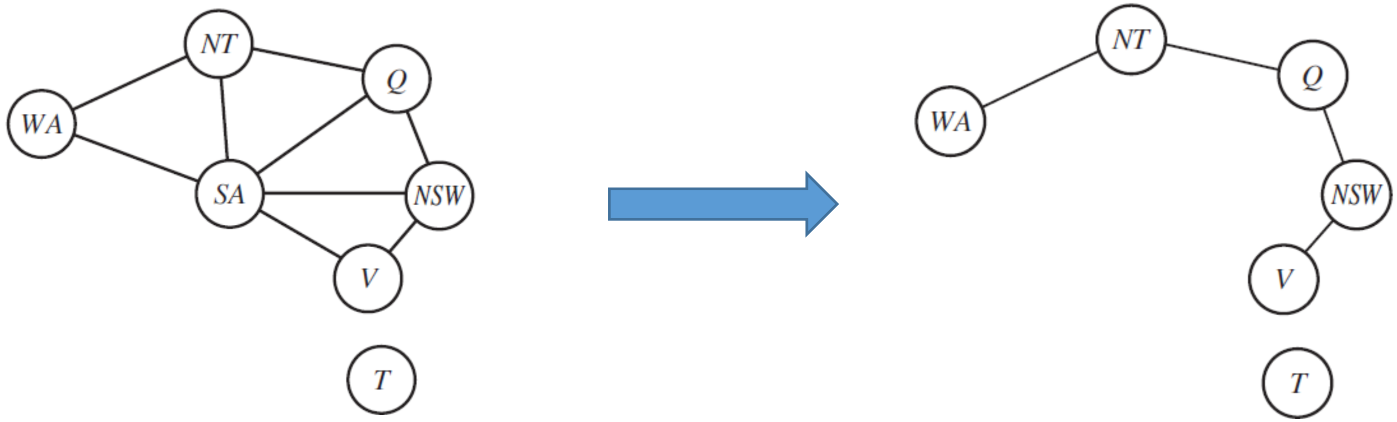


- No backtrack!

Nearly Tree-Structured CSP

- Reduce general constraint graphs to trees
  1. Removing nodes
  - Cutset conditioning: instantiate (in all ways) a set of variables such that the remaining constraint graph is a tree

1. Choose a subset S of the CSP's variables such that the constraint graph becomes a tree after removal of S. S is called a **cycle cutset**.
2. For each possible assignment to the variables in S that satisfies all constraints on S,
   (a) remove from the domains of the remaining variables any values that are inconsistent with the assignment for S, and
   (b) If the remaining CSP has a solution, return it together with the assignment for S.



Nearly Tree-Structured CSP

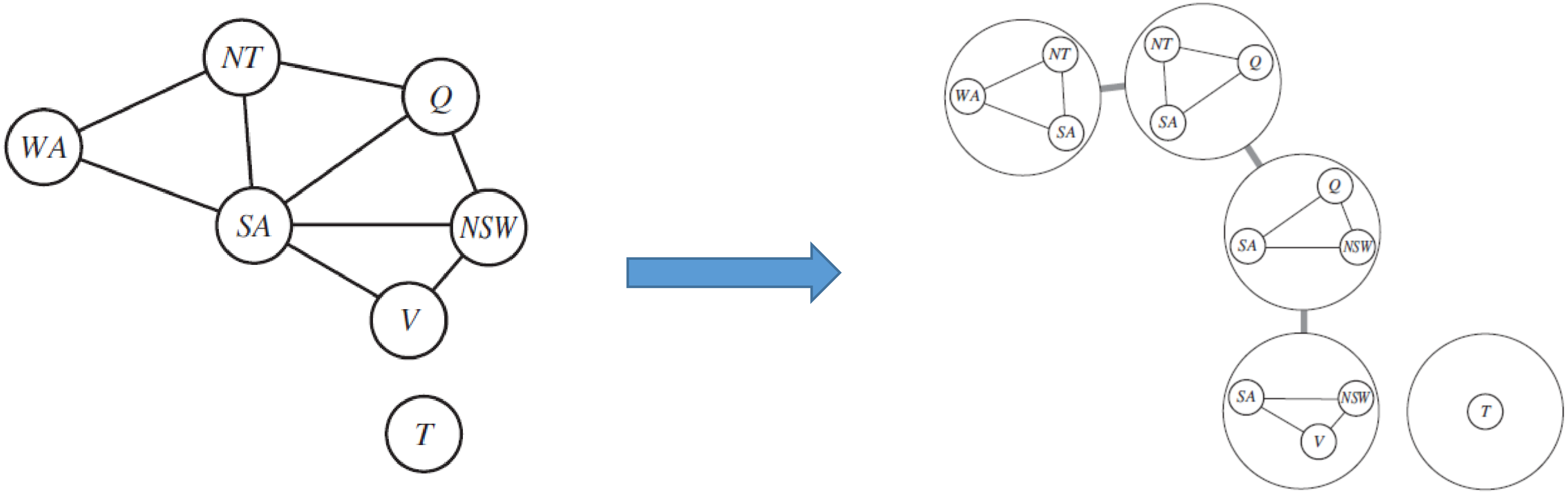
- Reduce general constraint graphs to trees
  1. Removing nodes
  - Cutset conditioning: instantiate (in all ways) a set of variables such that the remaining constraint graph is a tree
  - Cutset size c gives runtime $O(d^c (n-c) d^2)$, very fast for small c

1. Choose a subset S of the CSP's variables such that the constraint graph becomes a tree after removal of S. S is called a **cycle cutset**.
2. For each possible assignment to the variables in S that satisfies all constraints on S,
   (a) remove from the domains of the remaining variables any values that are inconsistent with the assignment for S, and
   (b) If the remaining CSP has a solution, return it together with the assignment for S.

Nearly Tree-Structured CSP

- Reduce general constraint graphs to trees
  2. Collapsing nodes together
  - Tree decomposition: construct the constraint graph into a set of connected subproblems.

- Every variable in the original problem appears in at least one of the subproblems.
- If two variables are connected by a constraint in the original problem, they must appear together (along with the constraint) in at least one of the subproblems.
- If a variable appears in two subproblems in the tree, it must appear in every subproblem along the path connecting those subproblems.



Nearly Tree-Structured CSP

- Reduce general constraint graphs to trees
  2. Collapsing nodes together
  - View each subproblem as a “mega-variable” whose domain is the set of all solutions for the subproblem.
  - Solve the constraints connecting between subproblems using the algorithm for tree-structured CSPs.
  - The problem can be solved in $O(nd^{w+1})$, w is the tree width of a tree decomposition.

Assignments

- Reading assignment:
  - Ch. 6.1-6.3, 6.5