CS3243: Introduction to Artificial Intelligence

Semester 2, 2019/2020

Midterm – CS3243 2020

- Closed book examination;
- we provide a “cheat sheet” in the test body.
- **One** A4 size sheet with your own notes, written in reasonable font + NUS approved calculator.

Venue : To be Announced (several)

Time: 15:00 – 17:00 March 7th (Saturday) 2020.

Previously…

- Minimax algorithm
  - What if MIN plays sub-optimally and MAX plays MINIMAX strategy?
  - Generalize to non-zero-sum, multi-player game? See section 5.2.2 of AIMA 3rd Ed.
- α-β pruning algorithm
  - Prune subtrees that won't affect minimax decision
- H-Minimax algorithm
  - Use heuristic functions and impose depth limit
- Expected Minimax algorithm
  - Solve stochastic games

Constraint Satisfaction Problems

AIMA Chapter 6.1 – 6.4

Outline

- Constraint satisfaction problems (CSPs)
- Backtracking search for CSPs
- Local consistency in constraint propagation
- Local search for CSPs

CSP: a Broad Modelling Framework

Linear programming

- Variables are all rational values
- Linear constraints

Combinatorial optimization

- Graph problems: vertex cover, bipartite matching, minimum spanning tree…
- Operations Research: scheduling, bin packing, planning, task allocation
- Game theory and economics: social choice, market pricing, goods allocation…

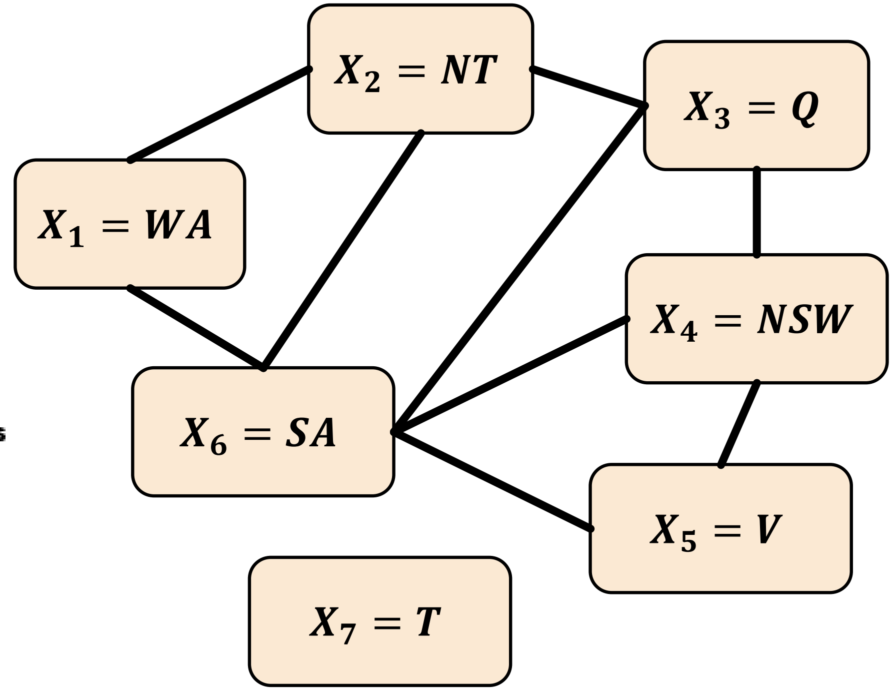
Effective CSP Solvers

- Variable dependencies reduce search space significantly
- Solve very large problem instances very quickly

Constraint Satisfaction Problems (CSPs)

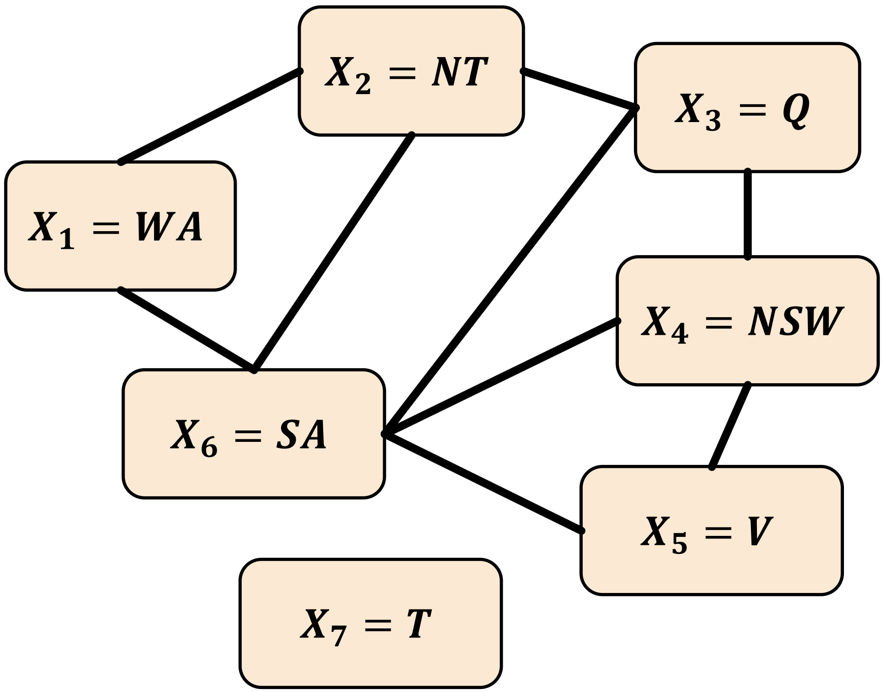
- Standard search problem:
  - state is atomic – any data structure that supports transition function, heuristic function, and goal test
- CSP
  - Variables $\vec{X} = X_1, \dots, X_n$; each variable X_i has a domain D_i
  - Constraints $\vec{C}$: what variable combinations are allowed?
    - Constraint scope: which variables are involved
    - Constraint relation: what is the relation between them.
  - Objective: find a legal assignment $(y_1, \dots, y_n)$ of values to variables
    - $y_i \in D_i$ for all $i \in [n]$
    - Constraints are all satisfied.

Example: Graph Coloring



Variables:	$\vec{X} = \langle WA, NT, Q, NSW, V, SA, T \rangle$
Domains:	$D_i = \{R, G, B\}$
Constraints:	If $(X_i, X_j) \in E$ then $color(X_i) \neq color(X_j)$

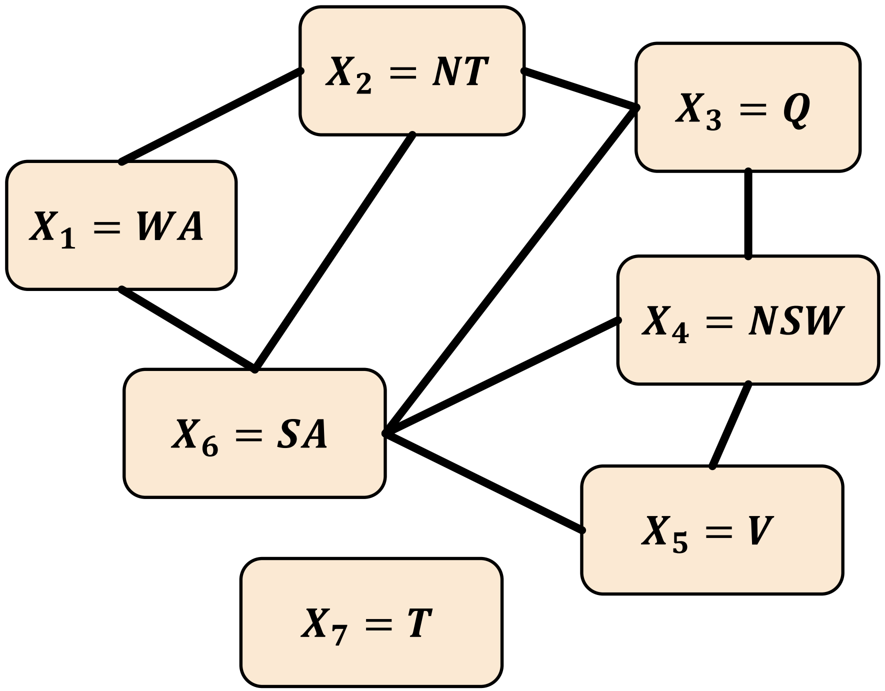
Example: Graph Coloring



Find a **complete**, **consistent** assignment

- **Complete:** all variables are set
- **Consistent:** no constraint is violated

Constraint Graph



- Binary CSP: each constraint relates two variables
- Constraint graph: nodes are variables, links are constraints

Example: Job-Shop Scheduling

- Car assembly consists of 15 tasks
- Variables: $Axle_F, Axle_B, Wheel_{LF}, Wheel_{RF}, Wheel_{LB}, Wheel_{RB}, Nuts_{LF}, Nuts_{RF}, Nuts_{LB}, Nuts_{RB}, Cap_{LF}, Cap_{RF}, Cap_{LB}, Cap_{RB}, Inspect$
- Domain: $D_i = \{1, 2, \dots, 27\}$
- Precedence constraints
  - e.g., $Axle_F + 10 \leq Wheel_{RF}$
- Disjunctive constraint
  - e.g., $(Axle_F + 10 \leq \ Axle_B)$ or $(Axle_B + 10 \leq Axle_F)$

CSP Variants

Discrete variables

- Finite domains:
  - n variables, domain size $d \rightarrow \mathcal{O}(d^n)$ complete assignments
  - e.g., 8-queens
- Infinite domains:
  - integers, strings, etc.
  - e.g., job-shop scheduling: variables are start/end days for each job
  - need a constraint language, e.g., $StartJob_3 + 5 \leq StartJob_4$

Continuous variables

- e.g., start/end times for Hubble Space Telescope observations
- Linear programming problems (i.e., with linear constraints) can be solved in polynomial time

Constraint Variants

Unary constraints

- Single variable: e.g., $SA \neq G$

Binary constraints

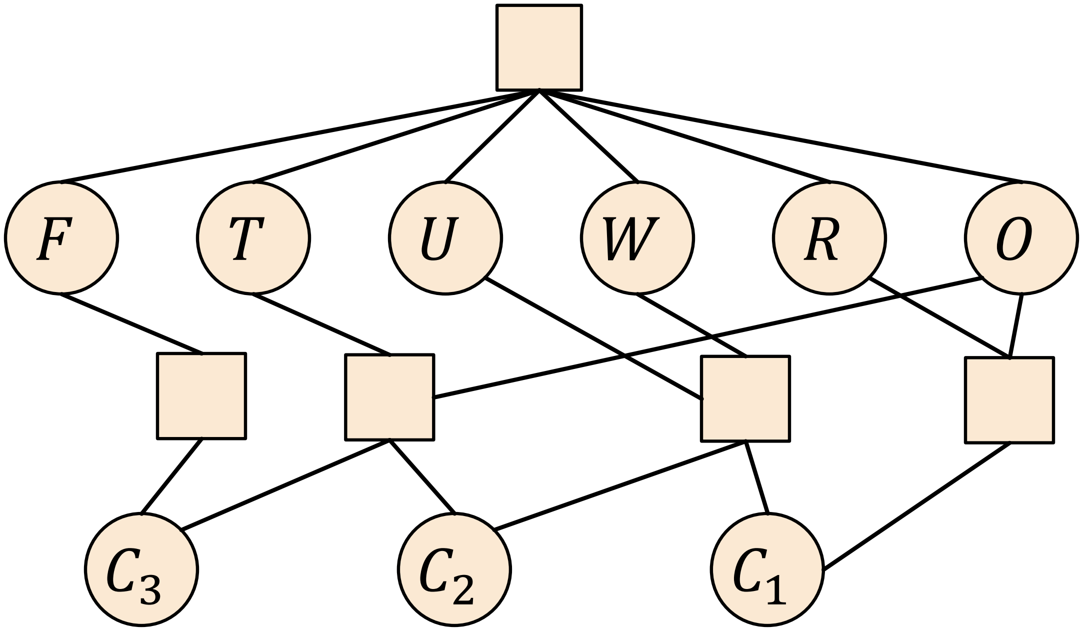
- e.g., *SA ≠ WA*

Global (i.e., higher-order) constraints

- 3 or more variables e.g. $X_1 + X_2 - 4X_7 \leq 15$

Cryptarithmetic Puzzles

$$\begin{array}{r} TWO \\ +TWO \\ \hline FOUR \end{array}$$



Variables:	$\vec{X} = \langle \mathrm{F, T, U, W, R, O, C_1, C_2, C_3} \rangle$
Domains:	$D_i = \{0, \ldots, 9\}$
Constraints:	$AllDiff(F, T, U, W, R, O)$ $O + O = R + 10C_1$ $C_1 + W + W = U + 10C_2$ $C_2 + T + T = O + 10C_3$ $C_3 = F$ $T, F \neq 0$

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)

Variables:	$A_1, \dots, A_9, \dots, I_1, \dots, I_9$ (81 variables)
Domains:	$D_i = \{1, \dots, 9\}$
Constraints:	$AllDiff(\dots) \times 27$ (9 columns, 9 rows, 9 boxes) e.g. $AllDiff(A_1, \dots, A_3; B_1, \dots, B_3; C_1, \dots, C_3)$ is the constraint for the top-left box.

Standard Search Formulation (Incremental)

- Each state is a partial variable assignment
- Initial state: the empty assignment []
- Transition function: assign a **valid** value to an unassigned variable → fail if no **valid** assignments
- Goal test: all variables have been assigned.

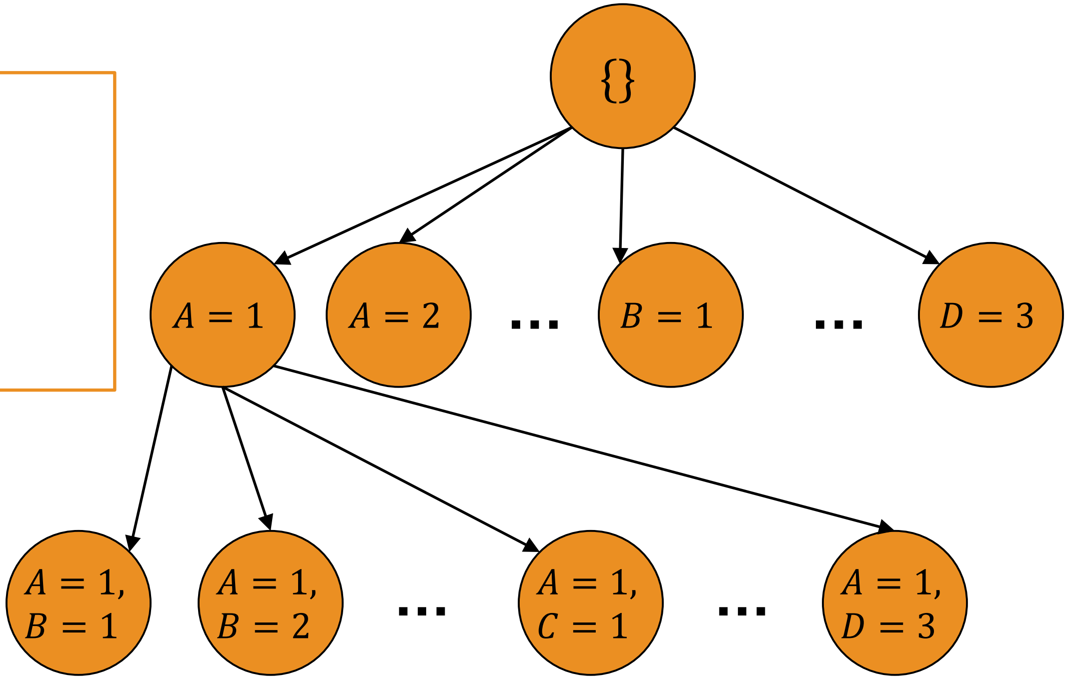
1. Uniform model for all CSPs; not domain-specific.
2. Every solution appears at depth n (n variables assigned)
3. Search path is irrelevant, can also use complete-state formulation.

What's the best search technique?

CSP Search Tree Size

CSP

- Variables: A, B, C, D
- Domains: $\{1,2,3\}$
- Assume no constraints



b at depth 0 = 4 variables $\times$ 3 values $=$ 12 states
b at depth 1 = 3 variables $\times$ 3 values $=$ 9 states
b at depth 2 = 2 variables $\times$ 3 values $=$ 6 states
b at depth 3 =1 variable $\times$ 3 values $=$ 3 states
At depth ℓ: $(n - \ell)d$

Total number of leaves = $n!\,d^n$

Backtracking Search

Order of variable assignment is irrelevant

- $[WA = R \text{ then } NT = G]$ equivalent to $[NT = G \text{ then } WA = R]$

At every level, consider assignments to a single variable.

- Fix an order of variable assignment → $b = d$ and there are d^n leaves

DFS for CSPs with single-variable/level assignments is called backtracking search

- Is the basic uninformed search algorithm for CSPs
- Backtracks when no legal assignments
- Can solve n-queens for $n \approx 25$

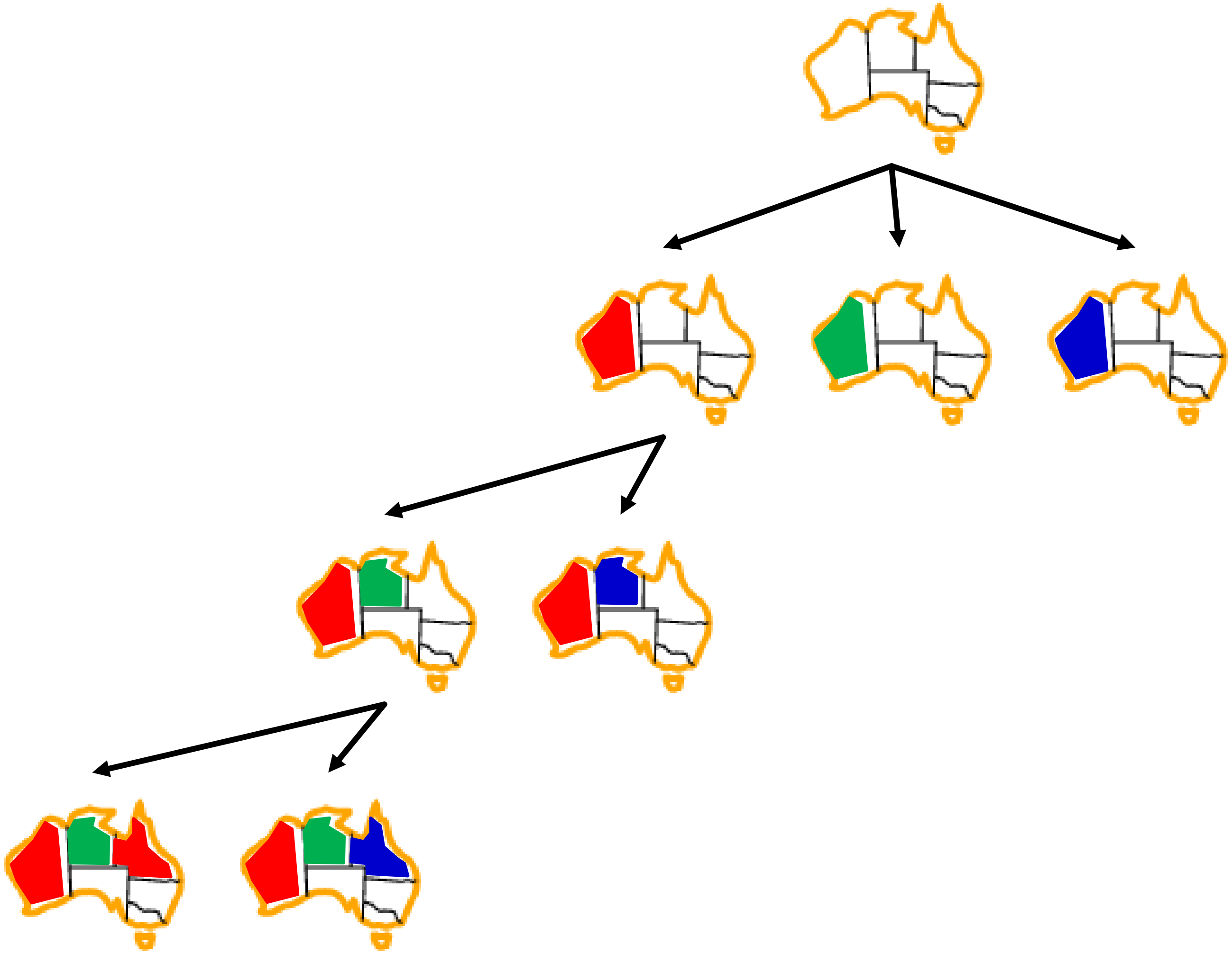
Backtracking Search

```
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 Example

Assign variables in the order of *WA*, *NT*, *Q*, …

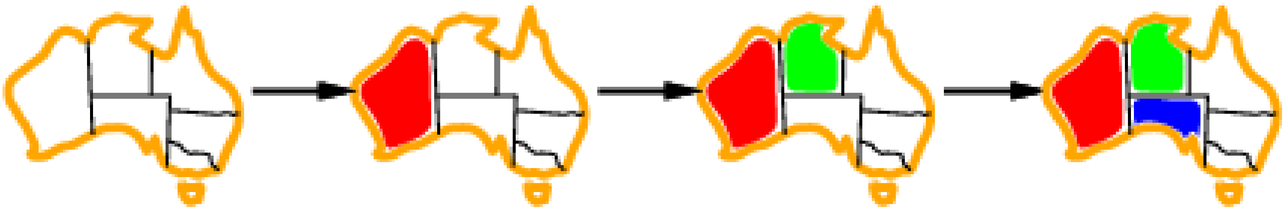


Improving Backtracking Efficiency

- General-purpose heuristics can give huge time gains:
  - SELECT-UNASSIGNED-VARIABLE: Which variable should be assigned next?
  - ORDER-DOMAIN-VALUES: In what order should its values be tried?
  - INFERENCE: How can domain reductions on unassigned variables be inferred?
  - Can we detect inevitable failure early?

Most Constrained Variable

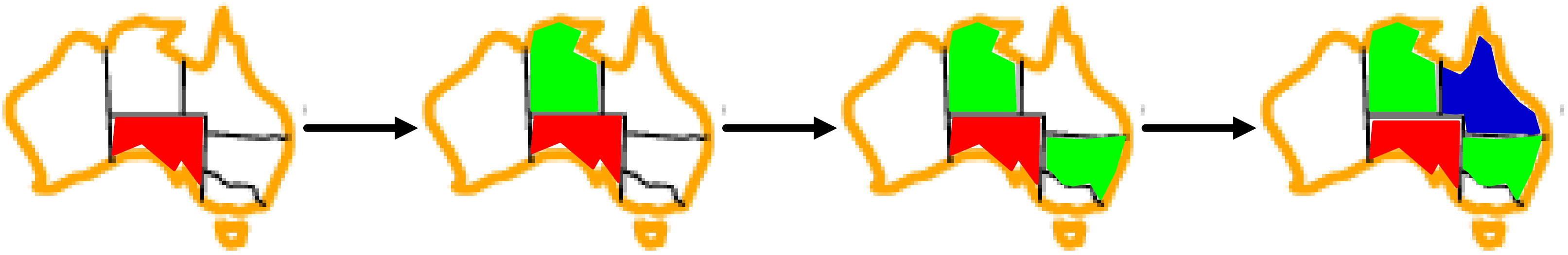
- Most constrained variable:
  - choose the variable with fewest legal values



- a.k.a. minimum-remaining-values (MRV) heuristic

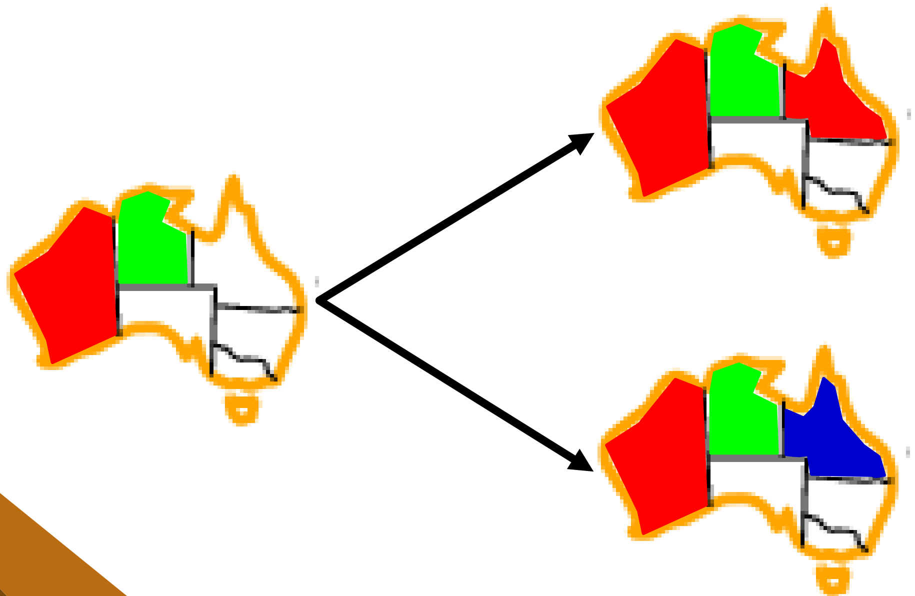
Most Constraining Variable

- Most constraining variable:
  - choose the variable with most constraints on remaining unassigned variables
- Tie-breaker among most constrained variables
- a.k.a. degree heuristic



Least Constraining Value

- Given a variable, choose the least constraining value:
  - the one that rules out the fewest values for neighboring unassigned variables



1 value for South Aust.

0 values for South Aust.

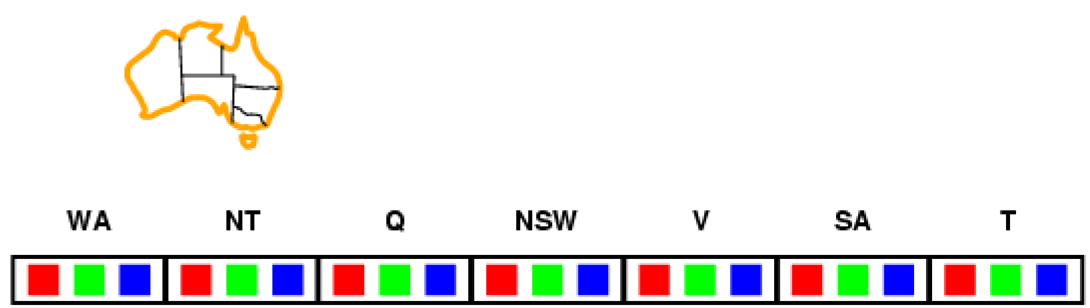
Inference in CSPs

Forward Checking and Arc Consistency

Forward Checking

Idea:

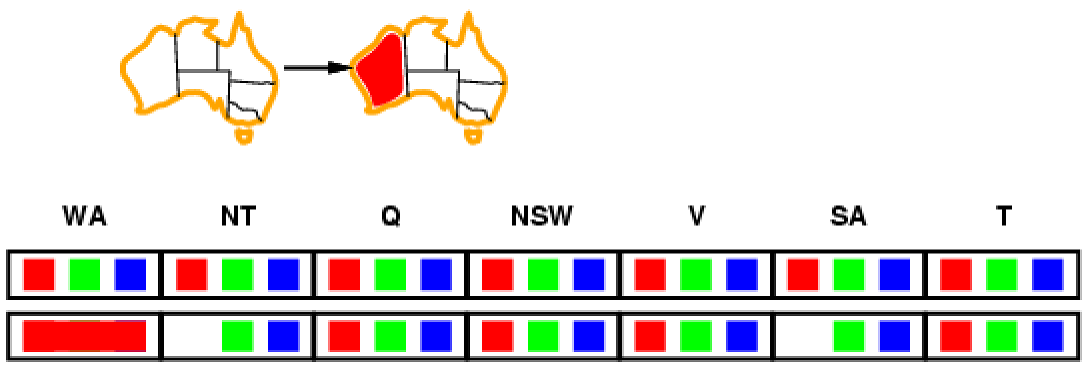
- Keep track of remaining legal values for unassigned variables
- Terminate search when any variable has no legal values



Forward Checking

Idea:

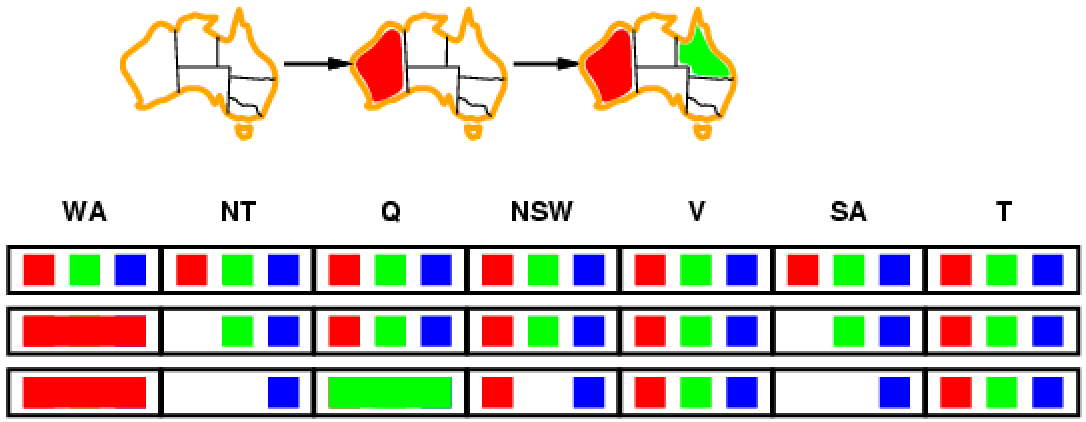
- Keep track of remaining legal values for unassigned variables
- Terminate search when any variable has no legal values



Forward Checking

Idea:

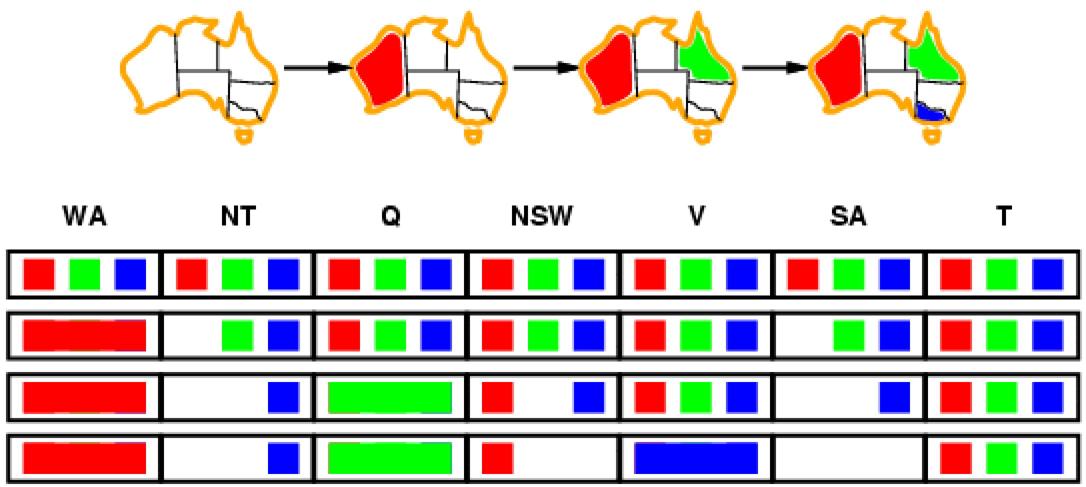
- Keep track of remaining legal values for unassigned variables
- Terminate search when any variable has no legal values



Forward Checking

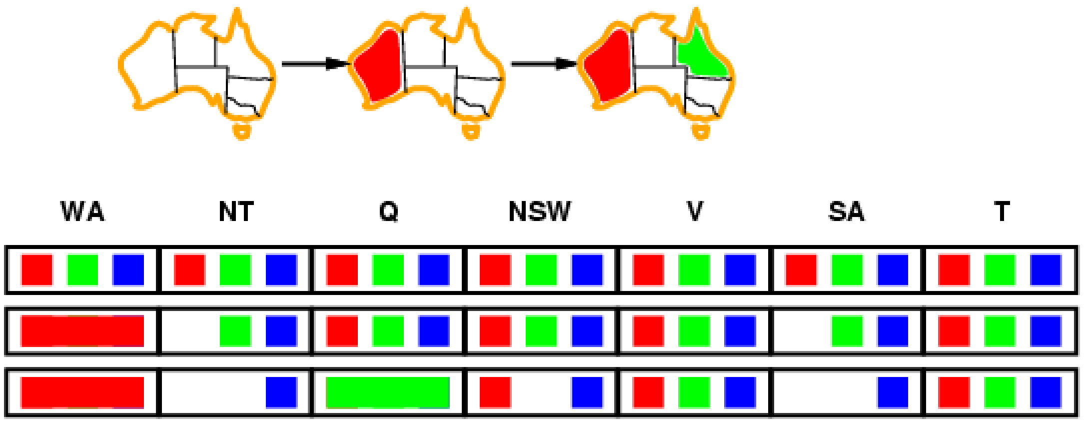
Idea:

- Keep track of remaining legal values for unassigned variables
- Terminate search when any variable has no legal values



Constraint Propagation

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

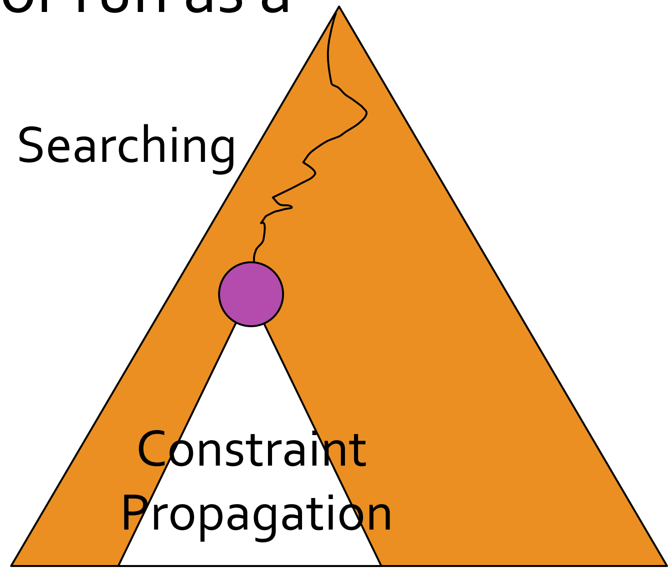


- NT and SA cannot both be blue!
- Solution: Constraint propagation repeatedly locally enforces constraints.

Inference in CSPs

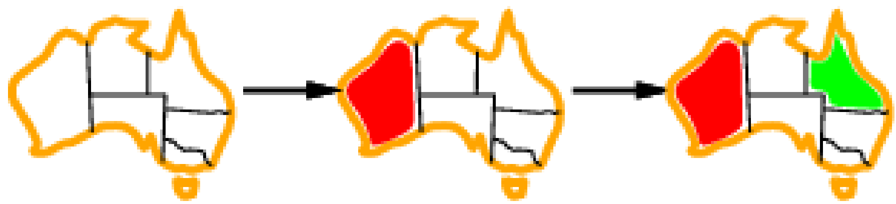
Try to infer illegal values for variables by performing constraint propagation

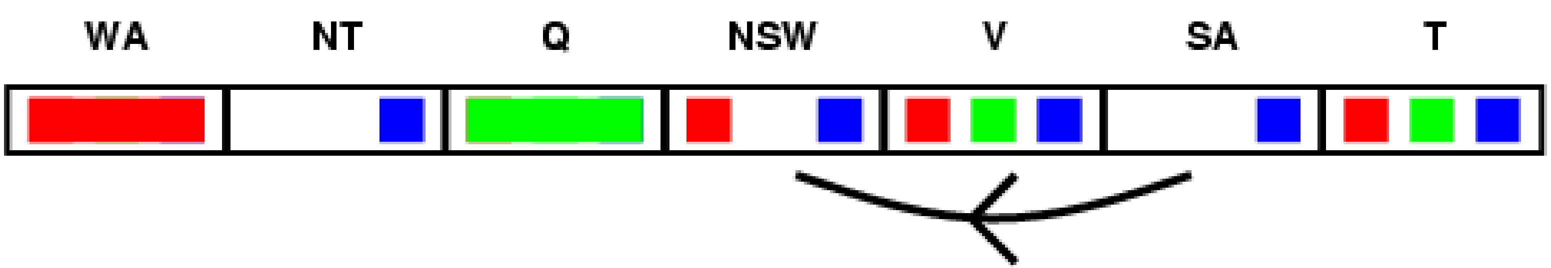
- Node consistency for unary constraints
- Arc consistency for binary constraints
- ...
- Can be interleaved with search, or run as a preprocessing step



Arc Consistency

X_i **is arc-consistent wrt** X_j **(the arc** (X_i, X_j) **is consistent) iff for every value** $x \in D_i$ **there exists some value** $y \in D_j$ **that satisfies the binary constraint on the arc** (X_i, X_j)



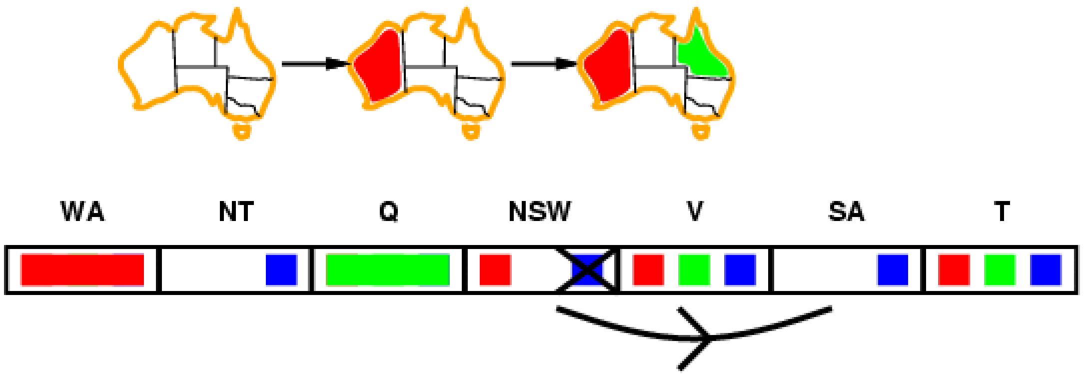


More on Arc Consistency

- If $X \leftarrow x$ makes a constraint impossible to satisfy, remove x from consideration!
- A value is impossible: the constraint provides ***no support*** for the value.
- E.g. $D_i = \{1,4,5\}, D_j = \{1,2,3\}$ and we have $X_i > X_j$; then $X_i \leftarrow 1$ is impossible.

Arc Consistency

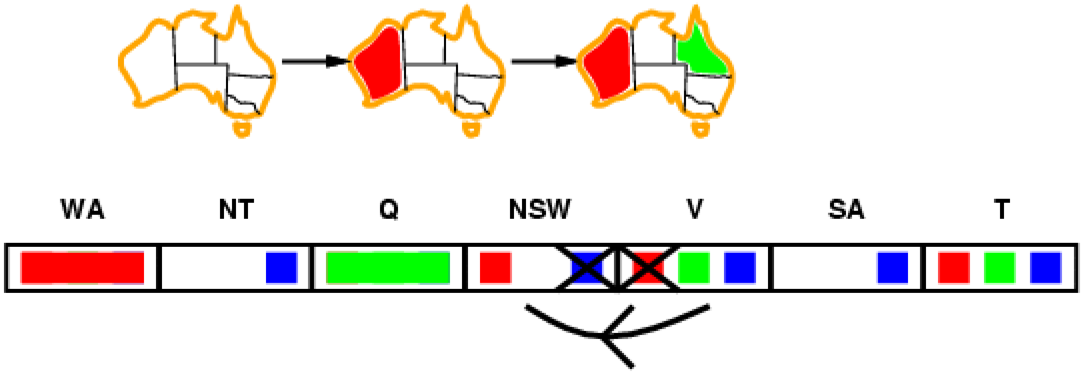
- Simplest form of propagation makes each arc consistent
- X_i is arc-consistent wrt X_j (the arc (X_i, X_j) is consistent) iff for every value $x \in D_i$ there exists some value $y \in D_j$ that satisfies the binary constraint on the arc (X_i, X_j)



- Arcs are directed, a binary constraint becomes two arcs
- Arc $NSW \rightarrow SA$ is originally not consistent, but is consistent after deleting NSW $=$ blue

Arc Consistency

- Simplest form of propagation makes each arc consistent
- X_i is arc-consistent wrt X_j (the arc (X_i, X_j) is consistent) iff for every value $x \in D_i$ there exists some value $y \in D_j$ that satisfies the binary constraint on the arc $\left(X_i, X_j\right)$



- If X_i loses a value, neighbors of X_i need to be (re)checked
- Arc $V \rightarrow NSW$ is originally not consistent, but is consistent after deleting $V =$ red

Arc Consistency Propagation

- Reducing D_i may result in domain reductions for others.
- E.g. $D_1 = \{1, 4, 5\}$, $D_2 = \{1,2,3\}$, $D_3 = \{2, 3, 4, 5\}$
  - Constraints: $X_3 > X_1$, $X_1 > X_2$
  - Remove 1 from D_1 so $D_1 = \{4, 5\}$
  - No support for $X_3 = 2,3,4$
  - We can remove those values: $D_3 = \{5\} \Rightarrow X_3 = 5$
  - Before applying AC to arc (X_1, X_2), we cannot reduce D_3
- A chain reaction of domain reductions.

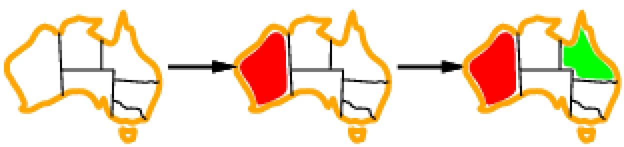
Sudoku Chain Reaction

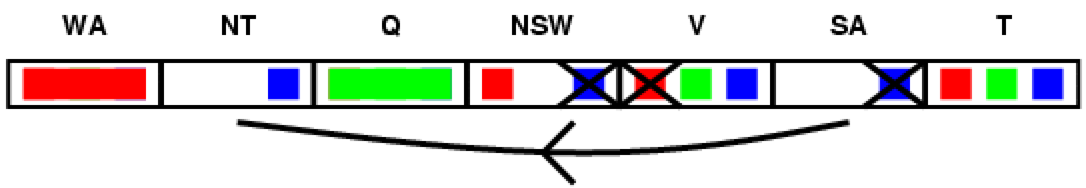
		3		2		6		
9			3		5			1
		1	8		6	4		
		8	1		2	9		
								8
		6	7		8	2		
		2	6		9	5		
8			2		3			9
		5		1		3		

- $Alldiff$ constraint on middle box makes domain of red square $\{3, 4, 5, 6, 9\}$. Column constraint reduces domain to $\{4\}$.
- Consider orange square. Original column and box constraints yield domain of $\{4,7\}$. Red square forces $\{7\}$.
- Blue box must be $\{1\}$ as column already has eight values.

Arc Consistency

- Simplest form of propagation makes each arc consistent
- X_i is arc-consistent wrt X_j (the arc (X_i, X_j) is consistent) iff for every value $x \in D_i$ there exists some value $y \in D_j$ that satisfies the binary constraint on the arc (X_i, X_j)





- Arc consistency propagation detects failure earlier than forward checking

Can be run as a preprocessing step or after each 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
      (Xi, Xj) ← REMOVE-FIRST(queue)
      if REVISE(csp, Xi, Xj) then
         if size of Di = 0 then return false
         for each Xk in Xi.NEIGHBORS - {Xj} do
            add (Xk, Xi) to queue
   return true
```

```
function REVISE( csp, Xi, Xj) returns true iff we revise the domain of Xi
   revised ← false
   for each x in Di do
      if no value y in Dj allows (x,y) to satisfy the constraint between Xi and Xj then
         delete x from Di
         revised ← true
   return revised
```

Time complexity: $\mathcal{O}(n^2 d^3)$

Time Complexity of AC-3

- CSP has at most n^2 directed arcs
- Each arc (X_i, X_j) can be inserted at most d times because X_i has at most d values to delete.
- REVISE: Checking the consistency of an arc takes $\mathcal{O}(d^2)$ time
- $\mathcal{O}(n^2 \times d \times d^2) = \mathcal{O}(n^2 d^3)$

```
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
      (Xi, Xj) ← REMOVE-FIRST(queue)
      if REVISE(csp, Xi, Xj) then
         if size of Di = 0 then return false
         for each Xk in Xi.NEIGHBORS - {Xj} do
            add (Xk, Xi) to queue
   return true

function REVISE( csp, Xi, Xj) returns true iff we revise the domain of Xi
   revised ← false
   for each x in Di do
      if no value y in Dj allows (x,y) to satisfy the constraint between Xi and Xj then
         delete x from Di
         revised ← true
   return revised
```

Maintaining AC (MAC)

- Like any other propagation, we can use AC in search
- i.e., search proceeds as follows:
  - establish AC at the root
  - when AC-3 terminates, choose a new variable/value
  - re-establish AC given the new variable choice (i.e. maintain AC)
  - repeat;
  - backtrack if AC gives empty domain
- Exception: initially, insert into queue only arcs of neighboring unassigned variables/nodes
- The hard part of implementation is undoing effects of AC

Special Types of Consistency

- Some types of constraints lend themselves to special types of arc-consistency
- Consider the $AllDiff$ constraint
  - the named variables must all take different values
  - not a binary constraint
  - can be expressed as $\frac{n(n-1)}{2}$ not-equal binary constraints
- We can apply, e.g., AC-3 as usual
- But there is a much better option

Generalized Arc Consistency and k-Consistency

- Suppose $D_1 = D_2 = \{2,3\}$ *and* $D_3 = \{1,2,3\}$.
- The constraints $X_i \neq X_j$ are arc-consistent
  - e.g., $X_2 = 2$ supports the value $X_3 = 3$
- The single ternary constraint $Alldiff(X_1, X_2, X_3)$ is not!
  - We must reduce $D_3 = \{1\}$.
- A special-purpose algorithm exists for *Alldiff* constraint to establish generalized arc consistency in efficient time
  - Special-purpose propagation algorithms are vital
- Arc Consistency (2-consistency) can be extended to k-consistency

Local Search for CSPs

- Hill-climbing, simulated annealing typically work with “complete” states, i.e., all variables assigned
- To apply to CSPs:
  - allow states that violate constraints
  - operators reassign variable values
- Variable selection: randomly select any conflicted variable
- Value selection by min-conflicts heuristic:
  - choose value that violates the fewest constraints
  - i.e., hill-climb with $h(n)$ = total # of violated constraints

Min-Conflicts

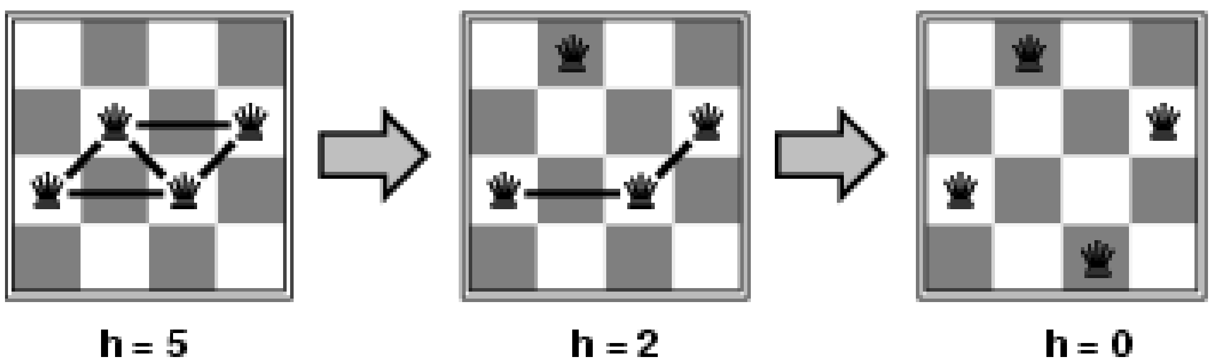
function MIN-CONFLICTS(csp, max_steps) **returns** a solution or failure
  inputs: csp, a constraint satisfaction problem
    max_steps, the number of steps allowed before giving up

  $current \leftarrow$ an initial complete assignment for csp
  for i = 1 to max_steps **do**
    if $current$ is a solution for csp **then return** $current$
    $var \leftarrow$ a randomly chosen conflicted variable from csp.VARIABLES
    $value \leftarrow$ the value v for var that minimizes CONFLICTS($var, v, current, csp$)
    set $var = value$ in $current$
  return $failure$

Figure 6.8 The MIN-CONFLICTS algorithm for solving CSPs by local search. The initial state may be chosen randomly or by a greedy assignment process that chooses a minimal-conflict value for each variable in turn. The CONFLICTS function counts the number of constraints violated by a particular value, given the rest of the current assignment.

Example: 4-Queens

- States: 4 queens in 4 columns ($4^4 = 256$ states)
- Actions: move queen in column
- Goal test: no attacks
- Evaluation: $h(n)$ = number of attacks

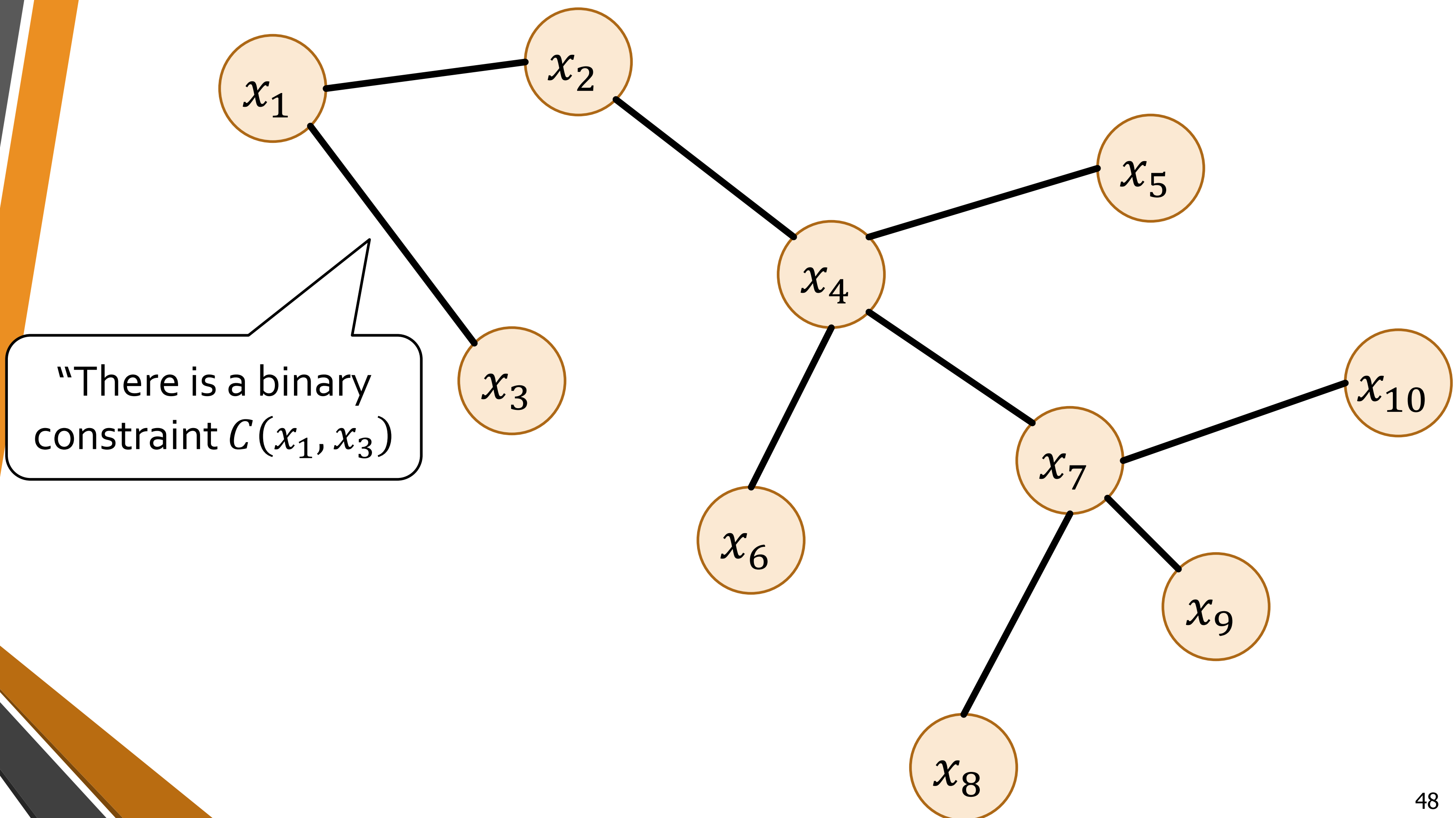


- Given random initial state, can solve n-queens in almost constant time for arbitrary n with high probability (e.g., $n = 10^6$)

Constraint Satisfaction Problems – Structured CSPs

AIMA Chapter 6.5

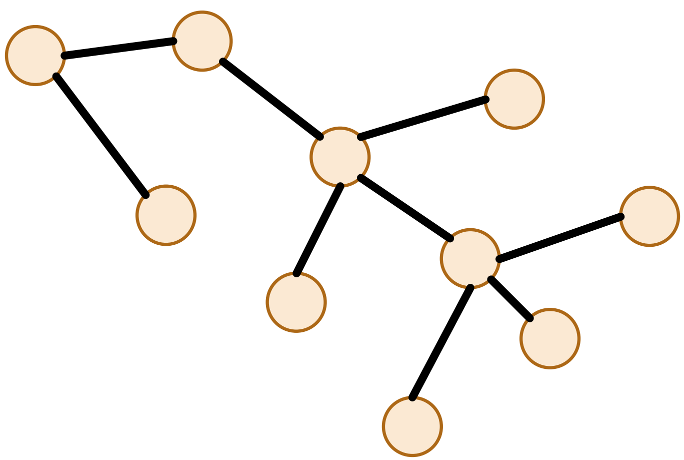
CSPs With Binary Constraints as Graphs



CSPs With Binary Constraints as Graphs

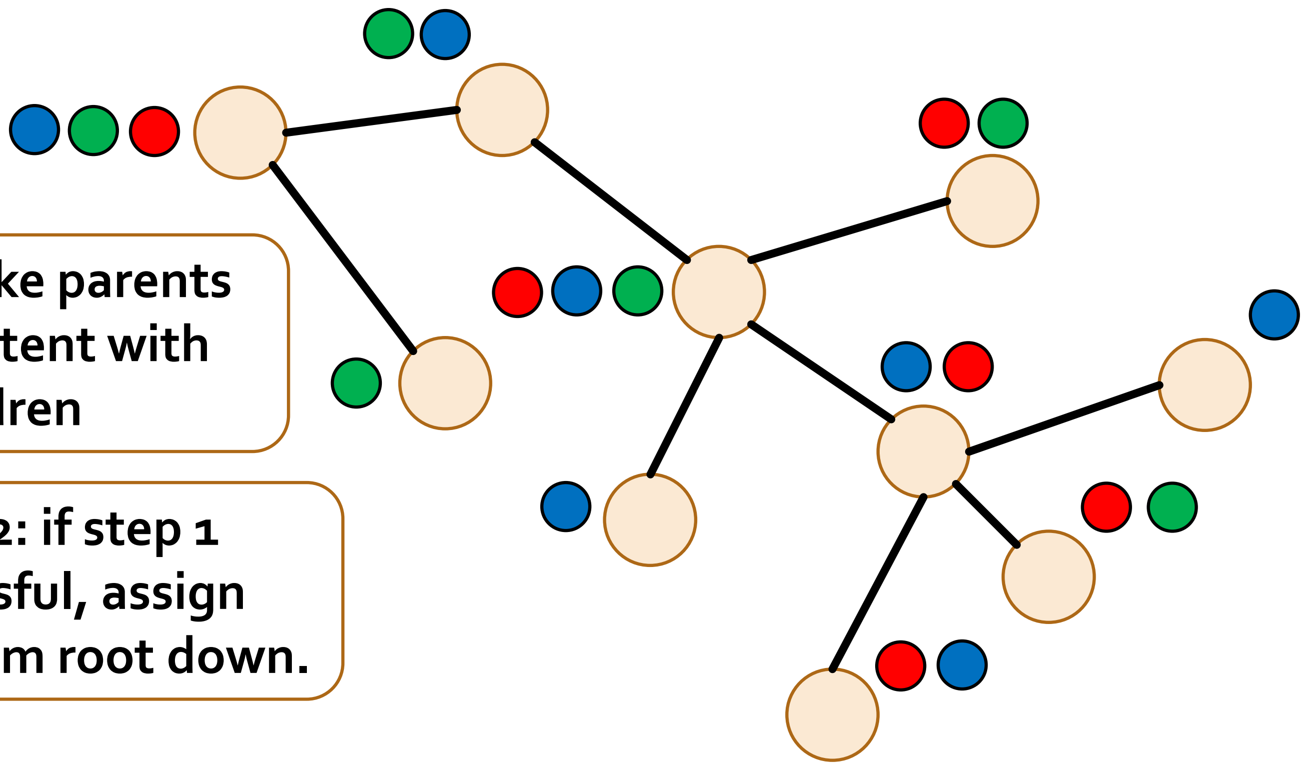
Theorem: if CSP constraint graph is a **tree**, then we can compute a satisfying assignment (or decide that one does not exist) in $\mathcal{O}(nd^2)$ time

No cycles!



CSPs With Binary Constraints as Graphs

Theorem: if CSP constraint graph is a **tree**, then we can compute a satisfying assignment (or decide that one does not exist) in $\mathcal{O}(nd^2)$ time

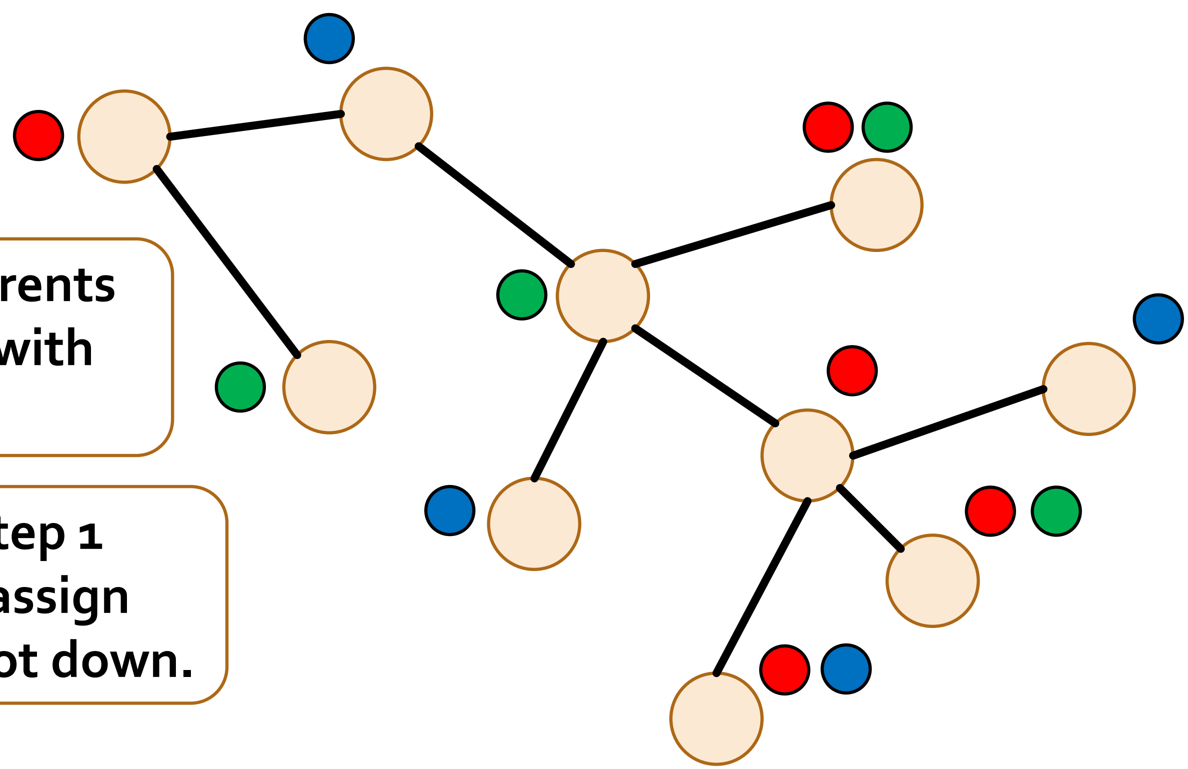


Step 1: make parents arc-consistent with children

Step 2: if step 1 successful, assign values from root down.

CSPs With Binary Constraints as Graphs

Theorem: if CSP constraint graph is a **tree**, then we can compute a satisfying assignment (or decide that one does not exist) in $\mathcal{O}(nd^2)$ time



Step 1: make parents arc-consistent with children

Step 2: if step 1 successful, assign values from root down.

CSPs With Binary Constraints as Graphs

Theorem: if CSP constraint graph is a **tree**, then we can compute a satisfying assignment (or decide that one does not exist) in $\mathcal{O}(nd^2)$ time

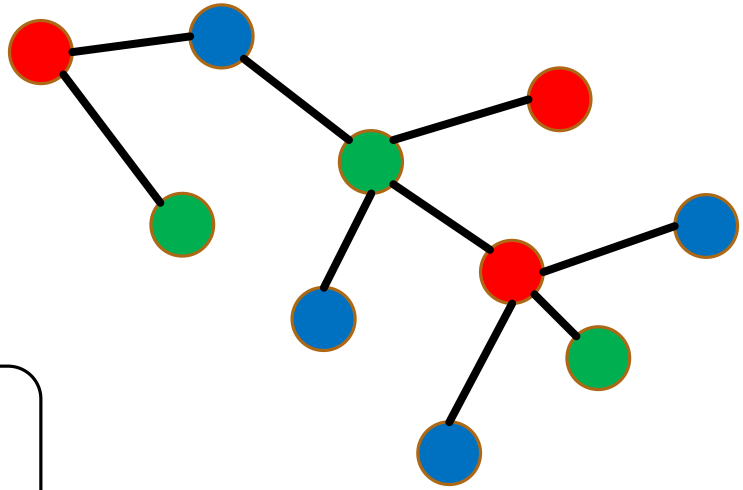
Topological sort takes $\boldsymbol{\mathcal{O}(n)}$ time on tree.

Step 1: make parents arc-consistent with children

n steps taking $\mathcal{O}(d^2)$ time each

Step 2: if step 1 successful, assign values from root down.

n steps taking $\mathcal{O}(d)$ time each

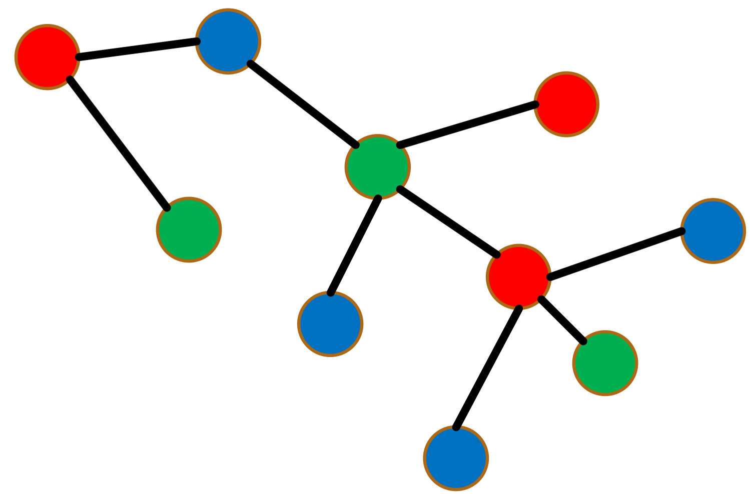


CSPs With Binary Constraints as Graphs

Theorem: if CSP constraint graph is a **tree**, then we can compute a satisfying assignment (or decide that one does not exist) in $\mathcal{O}(nd^2)$ time

Could it be that this step **removes options** for a **valid** CSP assignment? That would be very bad...

Step 1: make parents arc-consistent with children



Step 2: if step 1 successful, assign values from root down.

Because constraint graph is a tree, no need to backtrack when assigning! Assignment for node affects only its subtree.