SWINBURNE UNIVERSITY OF TECHNOLOGY

COS30018: Intelligent Systems

Problem-Solving Agents:

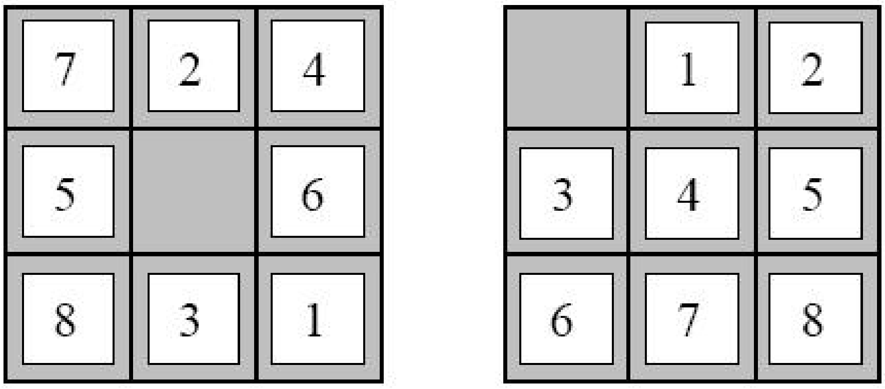
Search &

Constraint Satisfaction Problem

Outline

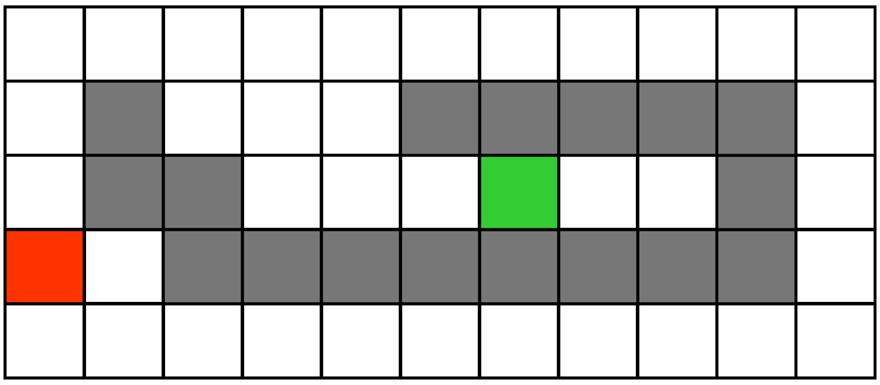
- Problem-solving agents
  - A kind of goal-based agent
- Problem formulation
  - Example problems
- Basic search algorithms
- Constraint Satisfaction Problems

Example: 8-puzzle



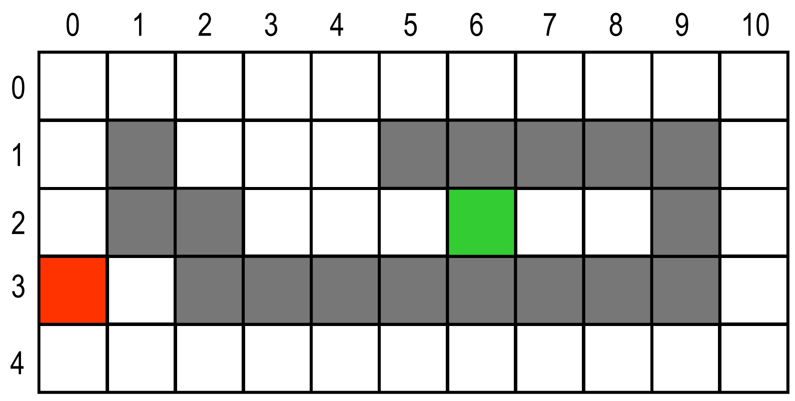
Start State Goal State

Example: Robot Navigation



🟥 Start state 🟩 Goal state

Example: Robot Navigation



Start state = [0, 3]

Goal state = [6,2]

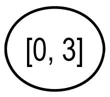
Single-state problem formulation

A problem is defined by four items:

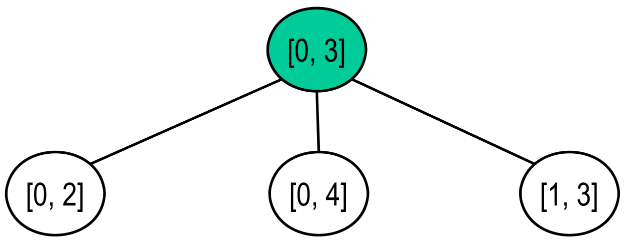
1. initial state e.g., [0, 3]
2. actions or successor function *S(x)* = set of action–state pairs
   - ☐ e.g., *S*([0,3]) = {<*UP*, [0, 2]>, <*DOWN*, [0, 4]>, <*RIGHT*, [1, 3]>}
3. goal test, can be
   - ☐ explicit, e.g., *x* = [6, 2]
   - ☐ implicit, e.g., *Checkmate(x)*
4. path cost (additive)
   - ☐ e.g., sum of distances, number of actions executed, etc.
   - ☐ *c(x,a,y)* is the step cost, assumed to be ≥ 0

- ■ A solution is a sequence of actions leading from the initial state to a goal state

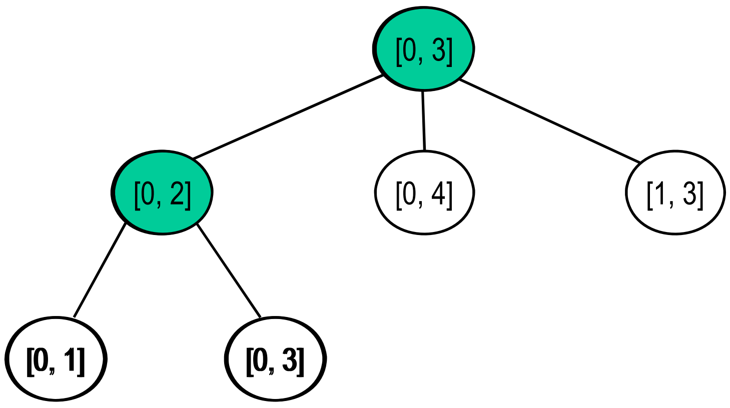
Simple tree search example



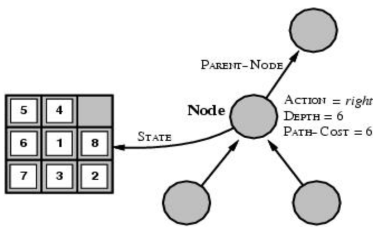
Simple tree search example

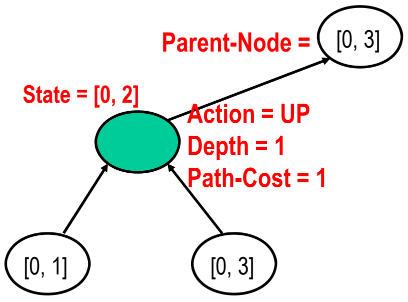


Simple tree search example



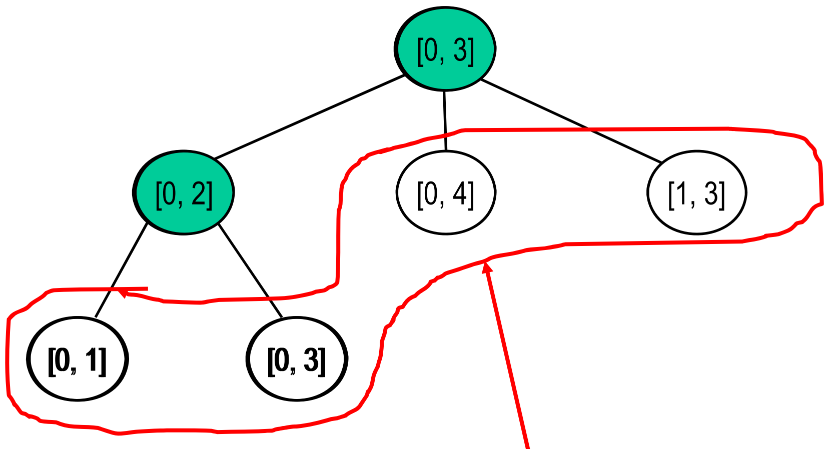
State space vs. search tree





- A *state* is a (representation of) a physical configuration
- A *node* is a data structure belong to a search tree
  - A node has a parent, children, … and includes path cost, depth, …
  - Here *node*= <*state, parent-node, action, path-cost, depth*>
  - *FRONTIER*= contains generated nodes which are not yet expanded.
    - White nodes with black outline

What is the *FRONTIER*?



FRONTIER= contains generated nodes which are not yet expanded.

Simple tree search algorithm

- By expanding the *FRONTIER*
  - According to a search strategy
  - Some popular search strategies: Depth-First Search (DFS), Breadth-First Search (BFS), Dijkstra Search (DS), Best-First Search such as Greedy Best First Search (GBFS) and A*
- The algorithm will systematically visit the states in the state space until it reaches a state that satisfies the goal test.
- In the process, the search tree is constructed in the computer memory
- When a goal state can be reached: The sequence of action steps leading from the initial state to that goal state can be reconstructed

  ➔ Solution is found and return
- Else:

  ➔ No Solution can be found.

Search where the path doesn't matter

- So far, we looked at problems where the path was the solution
  - ☐ Traveling on a graph
  - ☐ Eights puzzle
- However, in many problems, we just want to find a goal state
  - ☐ Doesn't matter how we get there

Queens puzzle

- Place eight queens on a chessboard so that no two attack each other

				Q			
		Q					
Q							
						Q	
	Q						
							Q
					Q		
			Q				

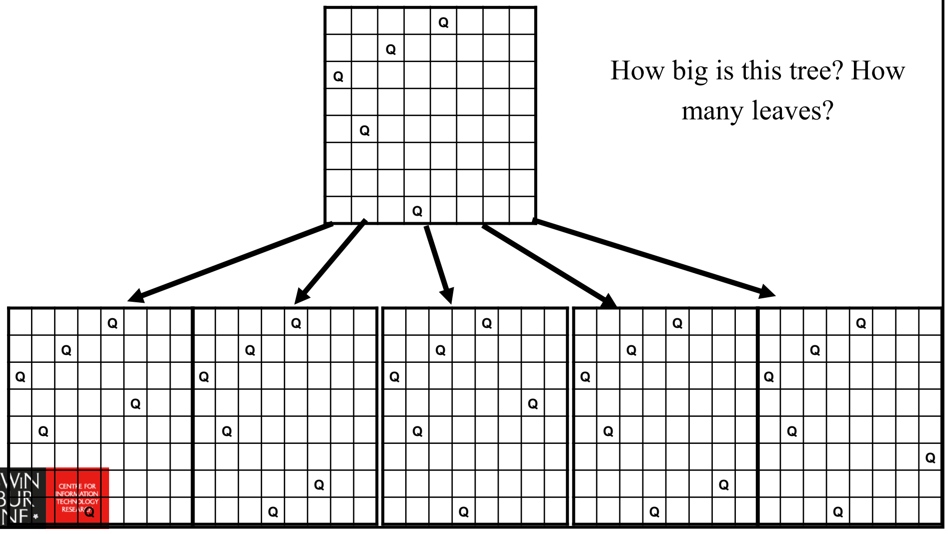
Sudoku

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

- **Each row, column and major block must be all different**
- **"Well posed" if it has unique solution: 27 constraints**

Search formulation of the queens puzzle

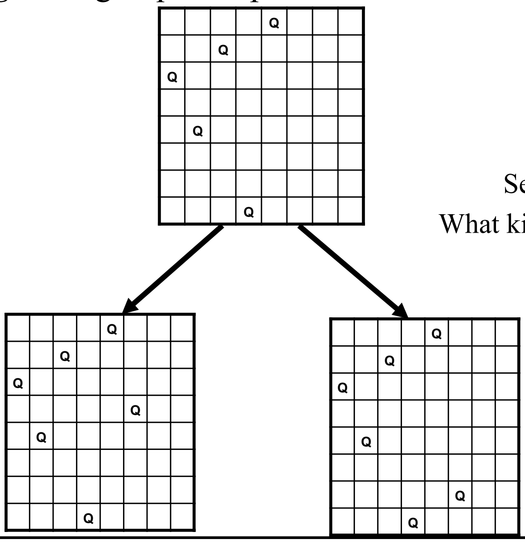
- **Successors**: all valid ways of placing additional queen on the board; **goal**: eight queens placed



How big is this tree? How many leaves?

Search formulation of the queens puzzle

- **Successors**: all valid ways of placing a queen in the next column; **goal**: eight queens placed

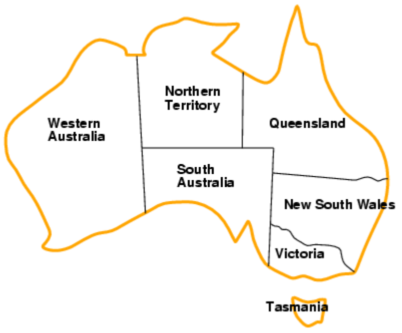


Search tree size?
What kind of search is best?

Constraint satisfaction problems (CSPs)

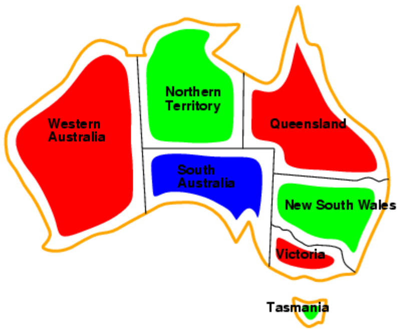
- Standard search problem:
  - ☐ state is a "black box“ – any data structure that supports successor function, heuristic function, and goal test
- CSP:
  - ☐ state is defined by variables X_i with values from domain D_i
  - ☐ goal test is a set of constraints specifying allowable combinations of values for subsets of variables
  - ☐ Simple example of a formal representation language
  - ☐ Allows useful general-purpose algorithms with more power than standard search algorithms

Example: Map-Coloring



- Variables *WA, NT, Q, NSW, V, SA, T*
- Domains D_i = {red,green,blue}
- Constraints: adjacent regions must have different colors
- e.g., WA ≠ NT, or (WA,NT) in {(red,green),(red,blue),(green,red), (green,blue),(blue,red),(blue,green)}

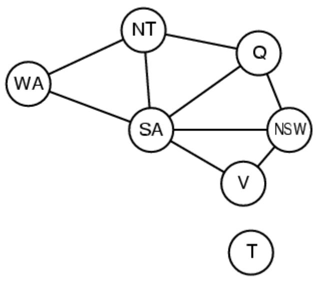
Example: Map-Coloring



- Solutions are complete and consistent assignments, e.g., WA = red, NT = green,Q = red,NSW = green,V = red,SA = blue,T = green

Constraint graph

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



Varieties of CSPs

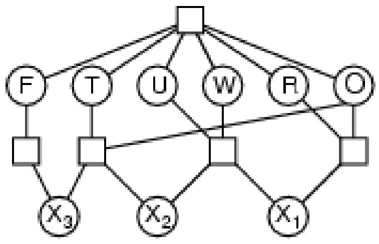
- Discrete variables
-
    - finite domains:
        - *n* variables, domain size *d* → $O(d^n)$ complete assignments
        - e.g., Boolean CSPs, incl.~Boolean satisfiability (NP-complete)
    - infinite domains:
        - integers, strings, etc.
        - e.g., job scheduling, variables are start/end days for each job
        - need a constraint language, e.g., $StartJob_1 + 5 \leq StartJob_3$
- Continuous variables
-
    - e.g., start/end times for Hubble Space Telescope observations
    - linear constraints solvable in polynomial time by linear programming

Varieties of constraints

- Unary constraints involve a single variable,
  - ☐ e.g., SA ≠ green
  - ☐
- Binary constraints involve pairs of variables,
  - ☐ e.g., SA ≠ WA
  - ☐
- Higher-order constraints involve 3 or more variables,
  - ☐ e.g., cryptarithmetic column constraints
  - ☐

Example: Cryptarithmetic

```
    T W O
  + T W O
  -------
  F O U R
```



- Variables: *F T U W R O X_1 X_2 X_3*
- Domains: *{0,1,2,3,4,5,6,7,8,9}*
  - ☐ Constraints:
  - ☐ *Alldiff (F,T,U,W,R,O)*
  - ☐ $O + O = R + 10 \cdot X_1$
  - ☐ $X_1 + W + W = U + 10 \cdot X_2$
  - ☐ $X_2 + T + T = O + 10 \cdot X_3$
  - ☐ $X_3 = F$, $T \neq 0$, $F \neq 0$

Real-world CSPs

- Staff assignment problems
  - e.g., who teaches what unit, which software developer does what tasks
- Timetabling problems
  - e.g., which class is offered when and where?
- Transportation scheduling
- Factory scheduling
- Notice that many real-world problems involve real-valued variables
-

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 does not conflict with current assignment
  $\rightarrow$ fail if no legal assignments
- Goal test: the current assignment is complete

1. This is the same for all CSPs
2. Every solution appears at depth *n* with *n* variables
   $\rightarrow$ use depth-first search
3. Path is irrelevant, so can also use complete-state formulation

Backtracking search

- Depth-first search for CSPs with single-variable assignments is called backtracking search
  - Backtracking is the idea that, as you go depth-first down a branch of the search tree and you try to find an assignment for the next variable that does not violate any constraint but you can't then you have to backtrack (from that current branch) to get to another branch
- Backtracking search is the basic **uninformed** algorithm for CSPs
- Can solve *n*-queens for $n \approx 25$
-

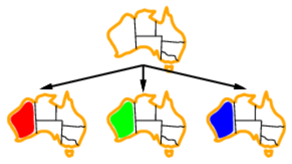
QUESTION: Can our search be **better informed**??

SWIN BUR NE CENTRE FOR INFORMATION TECHNOLOGY RESEARCH

Backtracking example

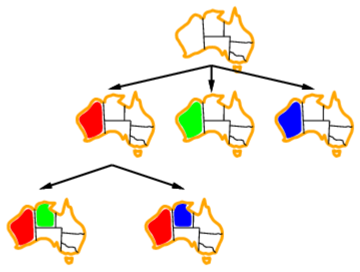
SWIN BUR NE CENTRE FOR INFORMATION TECHNOLOGY RESEARCH

Backtracking example



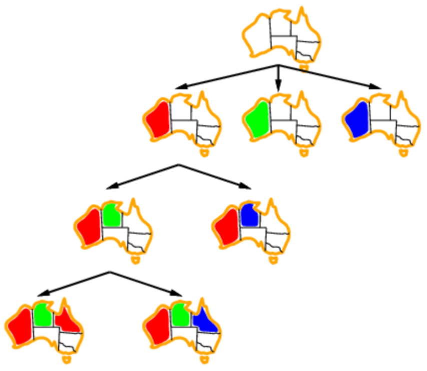
SWIN BUR * NE * CENTRE FOR INFORMATION TECHNOLOGY RESEARCH

Backtracking example

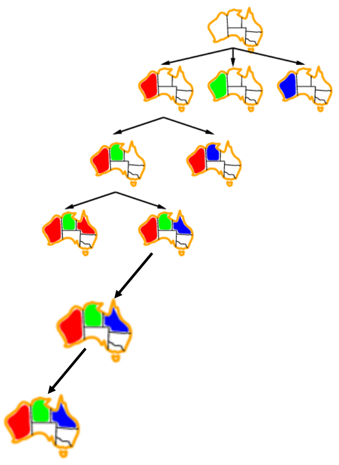


SWIN BUR * NE * CENTRE FOR INFORMATION TECHNOLOGY RESEARCH

Backtracking example



Backtracking example



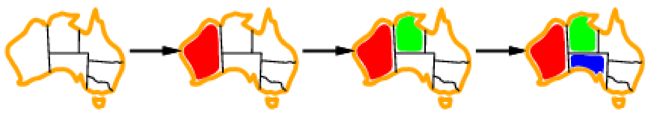
Improving backtracking efficiency

- General-purpose methods can give huge gains in speed (i.e. search in a smart way, **not in an uninformed manner**):
  - ☐ Which variable should be assigned next?
  - ☐
  - ☐ In what order should its values be tried?
  - ☐
  - ☐ Can we detect inevitable failure early?
  - ☐

Most constrained variable

- Most constrained variable:

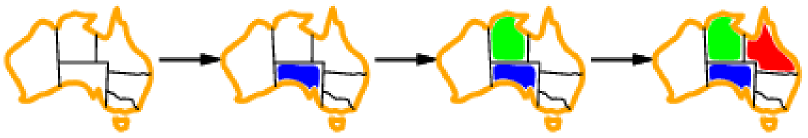
  choose the variable with the fewest legal values



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

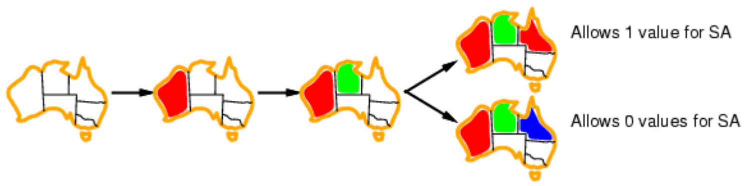
Most constraining variable

- ■ Tie-breaker among most constrained variables
- ■ Most constraining variable:
  - ☐ choose the variable with the most constraints on remaining variables
  - ☐

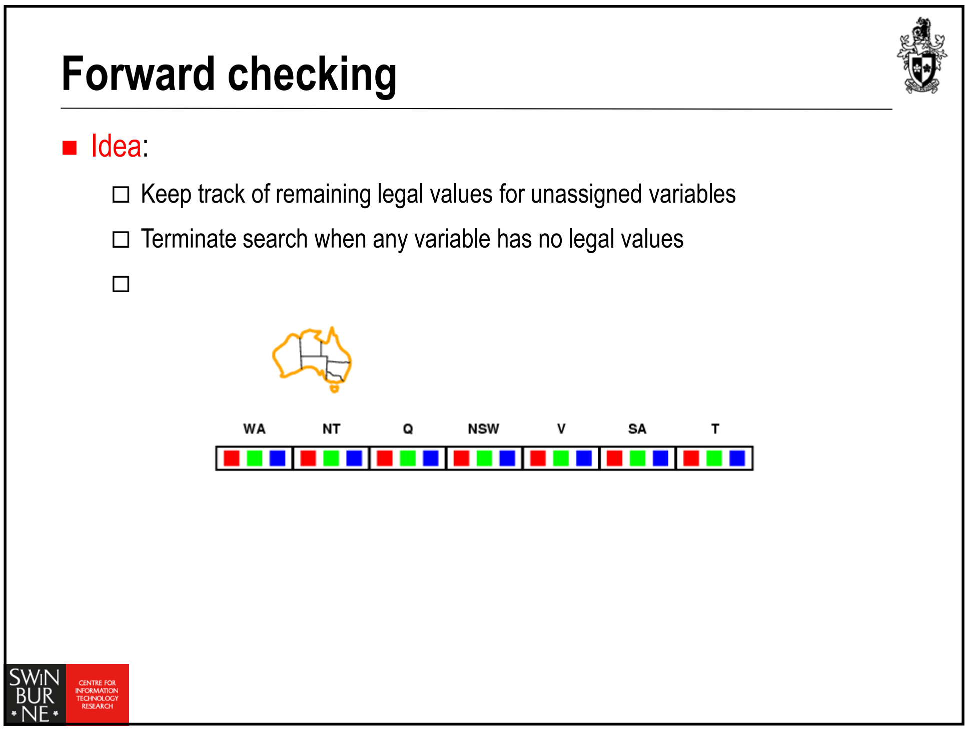


Least constraining value

- ■ Given a variable, choose the least constraining value:
- ■
  - ☐ the one that rules out the fewest values in the remaining variables
  - ☐



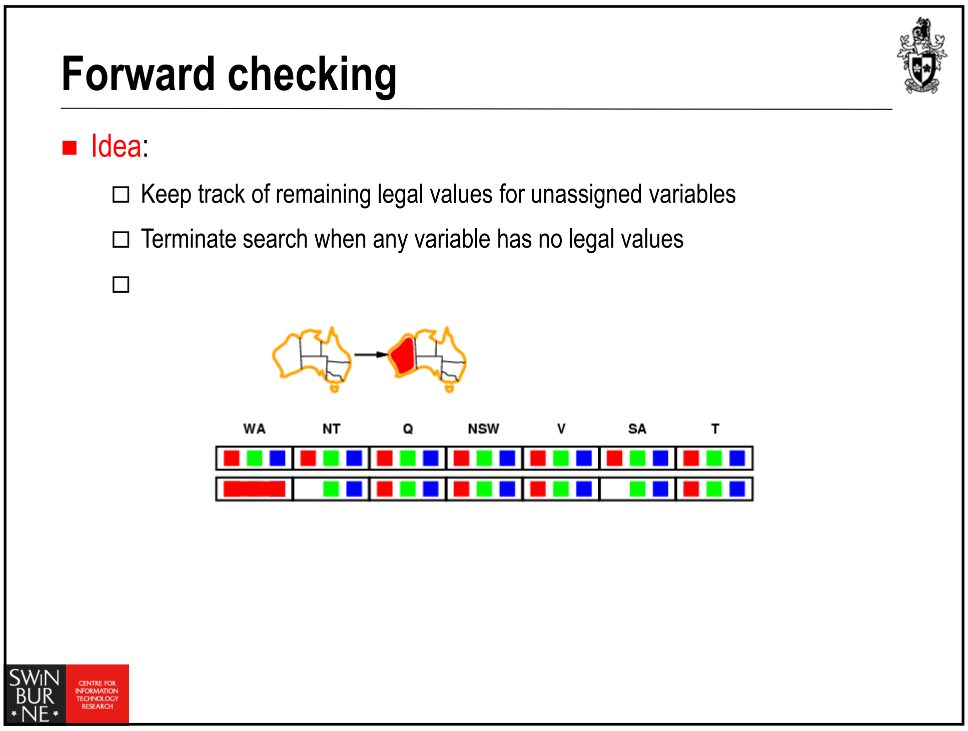
- ■ Combining these heuristics makes 1000 queens feasible



Forward checking

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

WA	NT	Q	NSW	V	SA	T
🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦



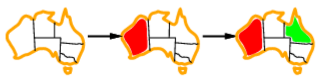
Forward checking

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

WA	NT	Q	NSW	V	SA	T
🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦
🟥	🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟩🟦	🟥🟩🟦

Forward checking

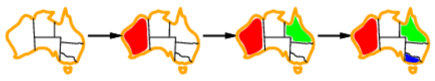
- Idea:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values
  -



WA	NT	Q	NSW	V	SA	T
🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦
🟥	🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟩🟦	🟥🟩🟦
🟥	🟦	🟩	🟥🟦	🟥🟩🟦	🟦	🟥🟩🟦

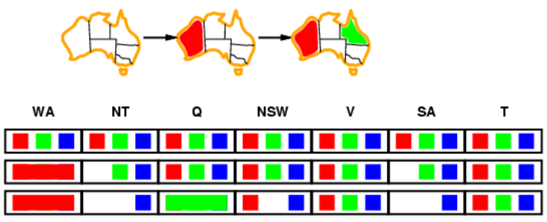
Forward checking

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



WA	NT	Q	NSW	V	SA	T
🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦
🟥	🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟩🟦	🟥🟩🟦
🟥	🟦	🟩	🟥🟦	🟥🟩🟦	🟦	🟥🟩🟦
🟥	🟦	🟩	🟥	🟦		🟥🟩🟦

Constraint propagation

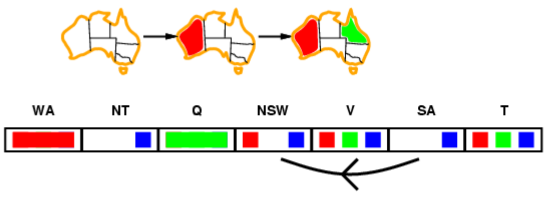
- Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures:
- 

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

Arc consistency

- Simplest form of propagation makes each arc consistent
- $X \rightarrow Y$ is consistent iff
-

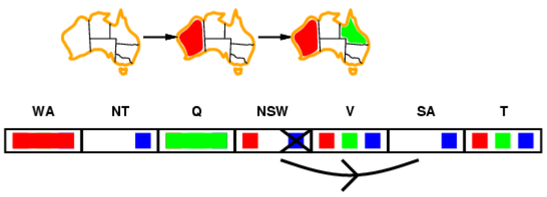
  for every value x of X there is some allowed y



Arc consistency

- Simplest form of propagation makes each arc consistent
- $X \rightarrow Y$ is consistent iff
-

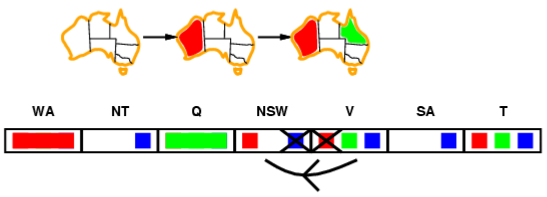
for every value x of X there is some allowed y



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

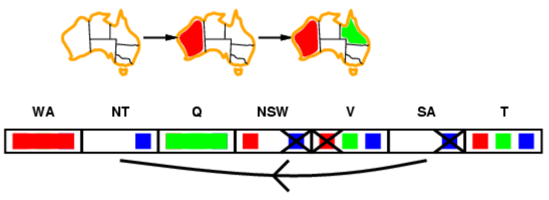


- If X loses a value, neighbors of X need to be rechecked
-

Arc consistency

- Simplest form of propagation makes each arc consistent
- *X* → *Y* is consistent iff

  for every value *x* of *X* there is some allowed *y*



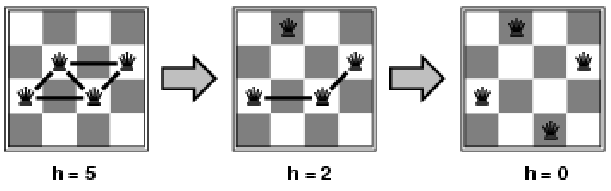
- If *X* loses a value, neighbors of *X* need to be rechecked
- Arc consistency detects failure earlier than forward checking
- Can be run as a preprocessor or after each assignment

Local search for CSPs

- Hill-climbing, simulated annealing typically work with "complete" states, i.e., all variables assigned
- To apply to CSPs:
  - ☐ allow states with unsatisfied 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 number of violated constraints
  - ☐

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* = 10,000,000)

Summary

- CSPs are a special kind of 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 variable assigned per node
- Variable ordering and value selection heuristics help significantly
- Forward checking prevents assignments that guarantee later failure
- Constraint propagation (e.g., arc consistency) does additional work to constrain values and detect inconsistencies
- Iterative min-conflicts is usually effective in practice