COMP3411: Artificial Intelligence

20. Constraint Satisfaction Problems

[Russell & Norvig: 5.1,5.2,5.3,4.3]

Outline

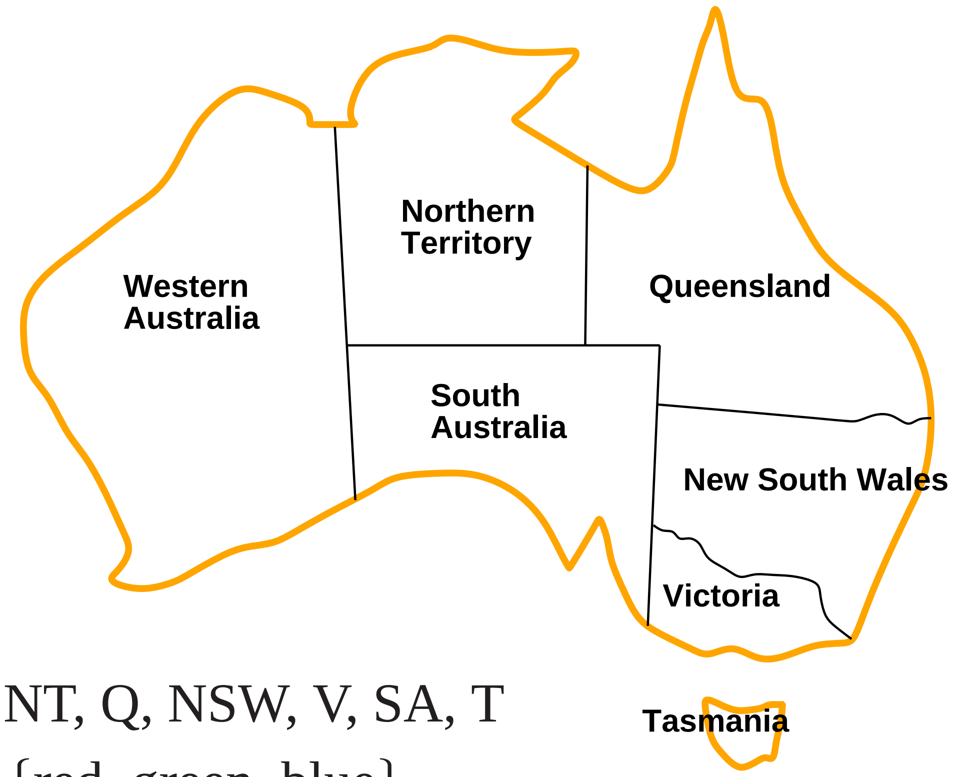
- Constraint Satisfaction Problems
- CSP examples
- backtracking search
- improvements to backtracking search
- local search
    - hill climbing
    - simulated annealing

Constraint Satisfaction Problems (CSPs)

Constraint Satisfaction Problems are defined by a set of variables X_i, each with a domain D_i of possible values, and a set of constraints C.

The aim is to find an assignment of the variables X_i from the domains D_i in such a way that none of the constraints C are violated.

Example: Map-Coloring



Variables WA, NT, Q, NSW, V, SA, T

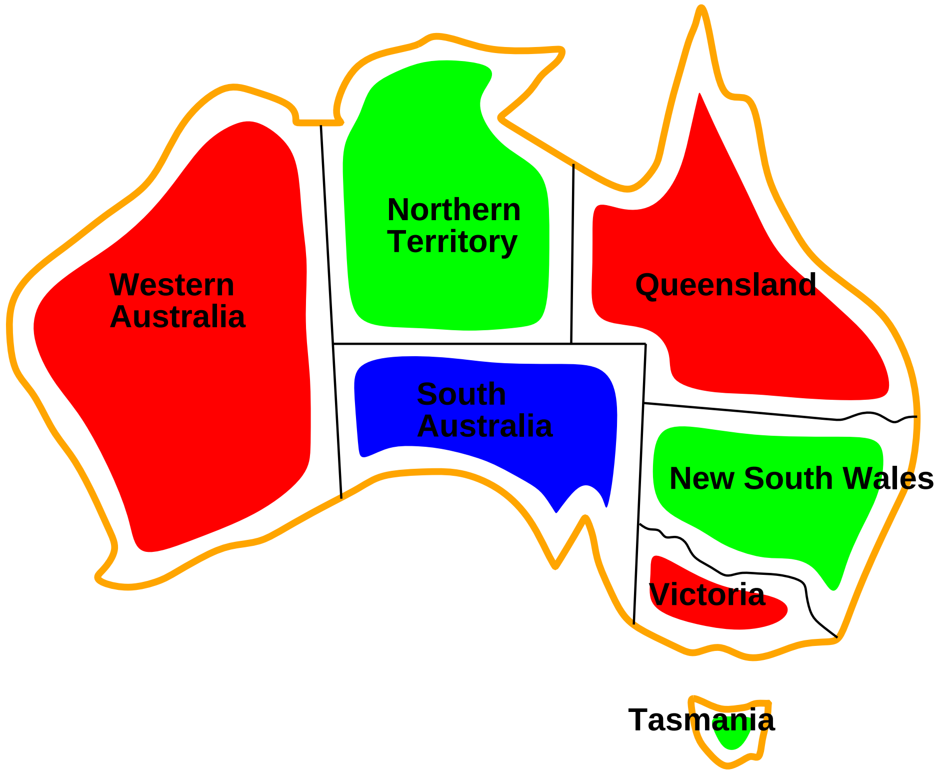
Domains $D_i = \{\text{red, green, blue}\}$

Constraints: adjacent regions must have different colors

e.g. WA$\neq$ NT, etc.

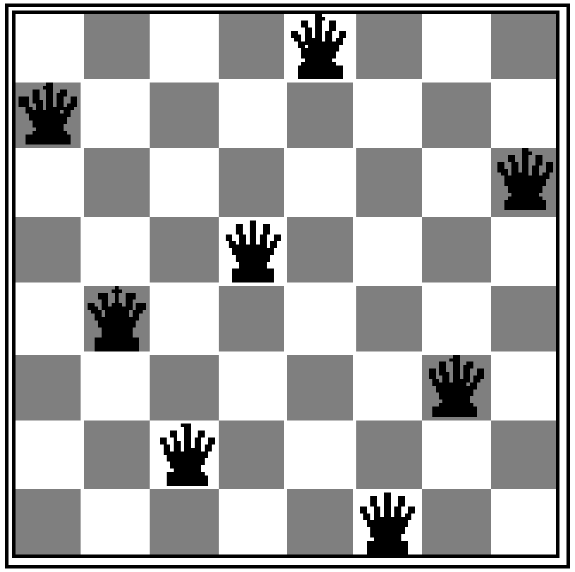
Example: Map-Coloring

Solution is an assignment that satisfies all the constraints, e.g.



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

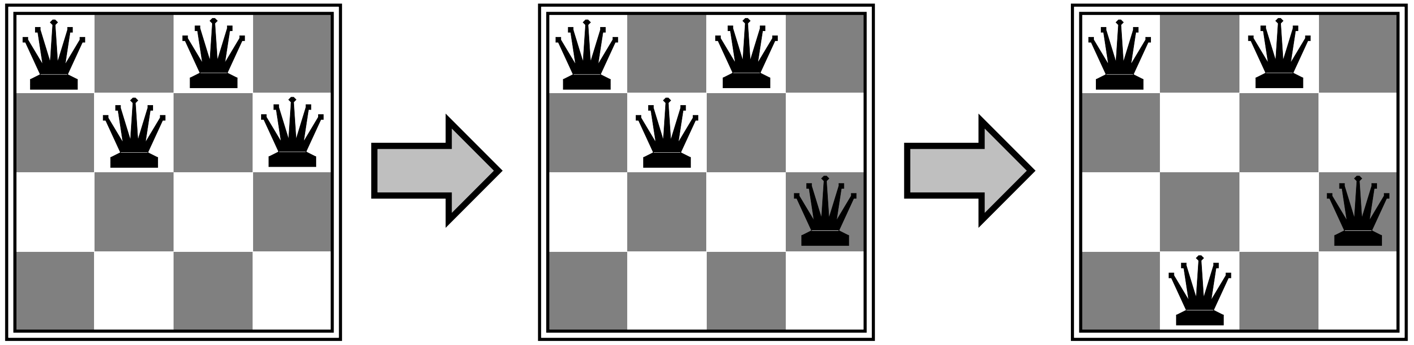
Example: n-Queens Puzzle



Put *n* queens on an *n*-by-*n* chess board so that no two queens are attacking each other.

n-Queens Puzzle as a CSP

Assume one queen in each column. Which row does each one go in?



Variables: Q_1, Q_2, Q_3, Q_4

Domains: $D_i = \{1,2,3,4\}$

Constraints:

$Q_i \neq Q_j$ (cannot be in same row)

$|Q_i - Q_j| \neq |i - j|$ (or same diagonal)

Example: Cryptarithmetic

$$\begin{array}{ccccc} & \mathrm{S} & \mathrm{E} & \mathrm{N} & \mathrm{D} \\ + & \mathrm{M} & \mathrm{O} & \mathrm{R} & \mathrm{E} \\ \hline \mathrm{M} & \mathrm{O} & \mathrm{N} & \mathrm{E} & \mathrm{Y} \end{array}$$

Variables:

D E M N O R S Y

Domains:

$\{0,1,2,3,4,5,6,7,8,9\}$

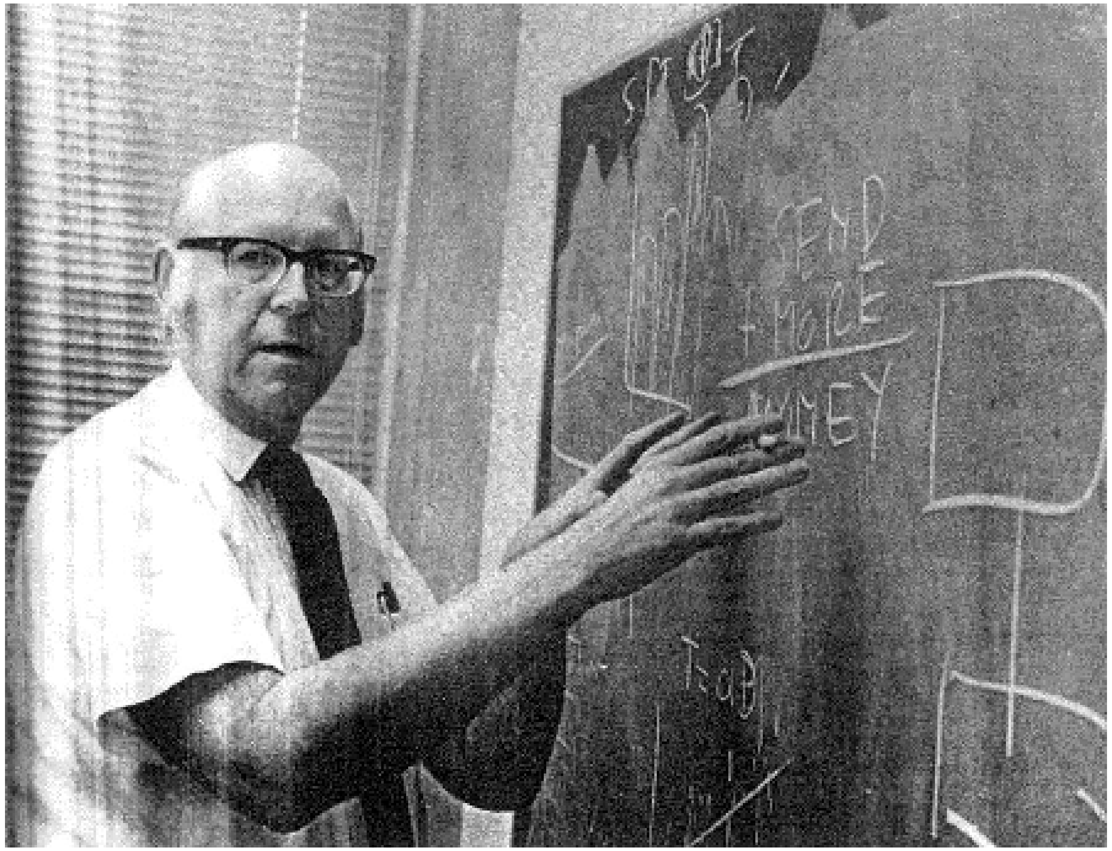
Constraints:

$M \neq 0$, $S \neq 0$ (unary constraints)

$Y = D + E$ or $Y = D + E - 10$, etc.

$D \neq E$, $D \neq M$, $D \neq N$, etc.

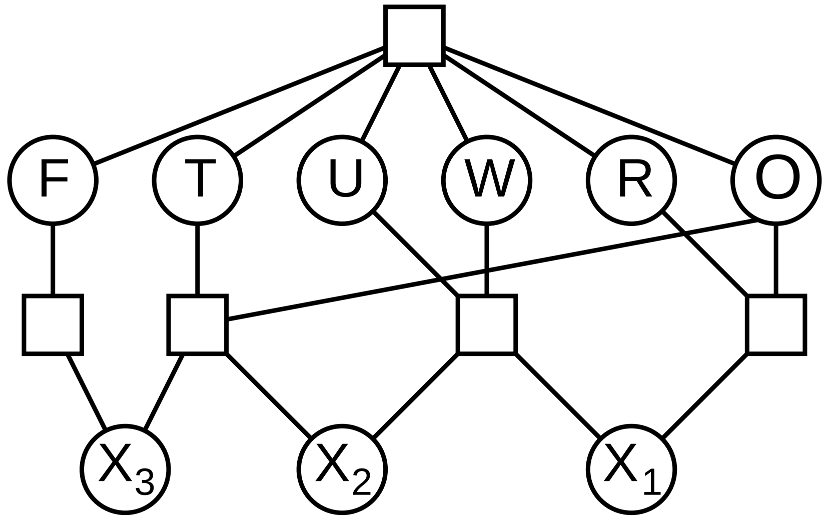
Cryptarithmetic with Allen Newell



Cryptarithmetic with Hidden Variables

We can add “hidden” variables to simplify the constraints.

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



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:

AllDifferent(F,T,U,W,R,O)

$O + O = R + 10 \cdot X_1$, etc.

Example: Sudoku

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

Real-world CSPs

- Assignment problems (e.g. who teaches what class)
- Timetabling problems (e.g. which class is offered when and where?)
- Hardware configuration
- Transport scheduling
- Factory scheduling

Varieties of constraints

- Unary constraints involve a single variable
  - M $\neq$ 0
- Binary constraints involve pairs of variables
  - SA $\neq$ WA
- Higher-order constraints involve 3 or more variables
  - $Y = D + E$ or $Y = D + E - 10$
- Inequality constraints on Continuous variables
  - $\text{EndJob}_1 + 5 \leq \text{StartJob}_3$
- Soft constraints (Preferences)
  - 11am lecture is better than 8am lecture!

Standard search formulation

Let’s start with a simple but slow approach, then see how to improve 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 previously assigned variables $\Rightarrow$ fail if no legal assignments (not fixable!)
- Goal test: the current assignment is complete

1) This is the same for all CSPs

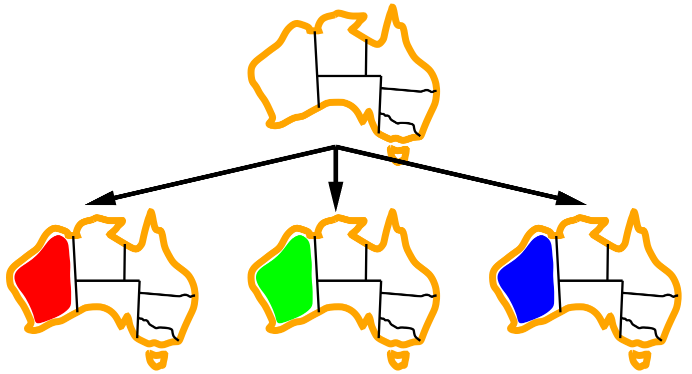
2) Every solution appears at depth n with n variables

$\Rightarrow$ use depth-first search

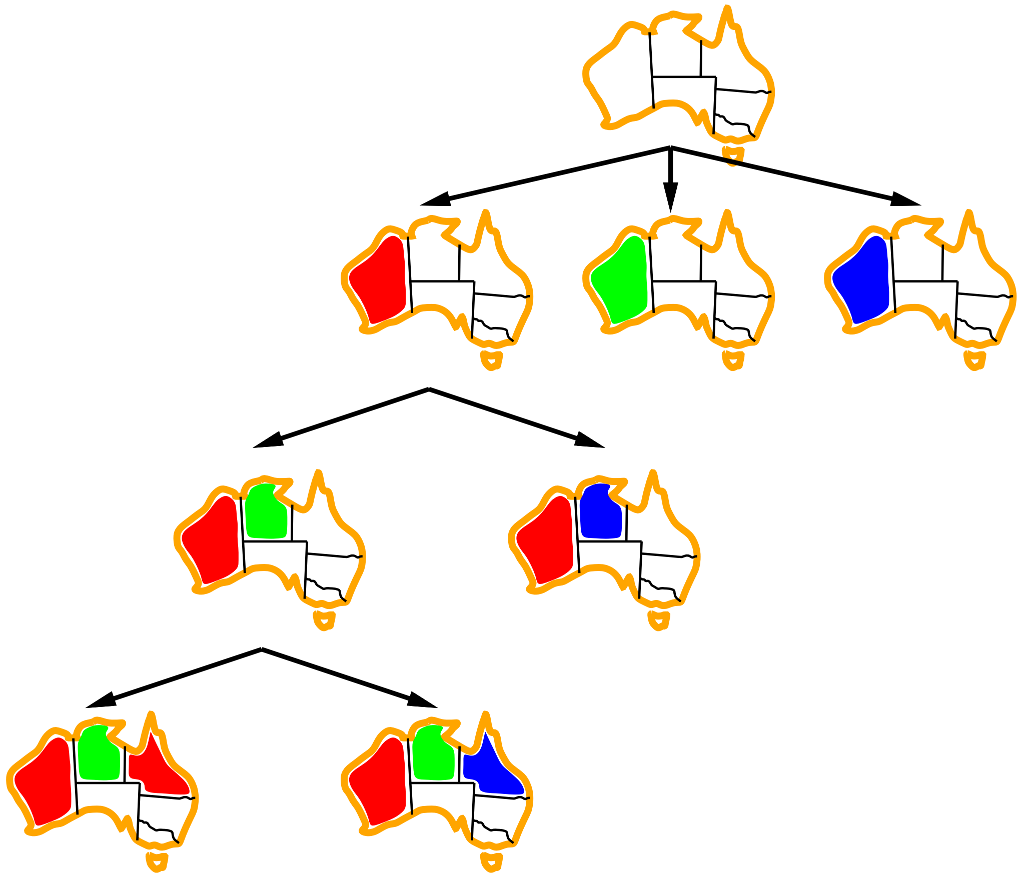
Backtracking search

- variable assignments are commutative
  [WA = red then NT = green] same as [NT = green then WA = red]
- only need to consider assignments to a single variable at each node
- depth-first search for CSPs with single-variable assignments is called Backtracking search
- Backtracking search is the basic algorithm for CSPs
- can solve n-queens for $n \approx 25$

Backtracking example



Backtracking example



Improvements to Backtracking search

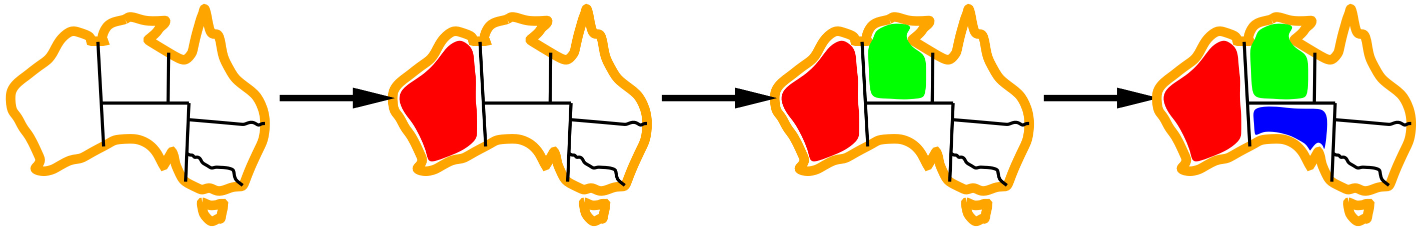
General-purpose heuristics can give huge gains in speed:

1. which variable should be assigned next?
2. in what order should its values be tried?
3. can we detect inevitable failure early?

Minimum Remaining Values

Minimum Remaining Values (MRV):

Choose the variable with the fewest legal values.

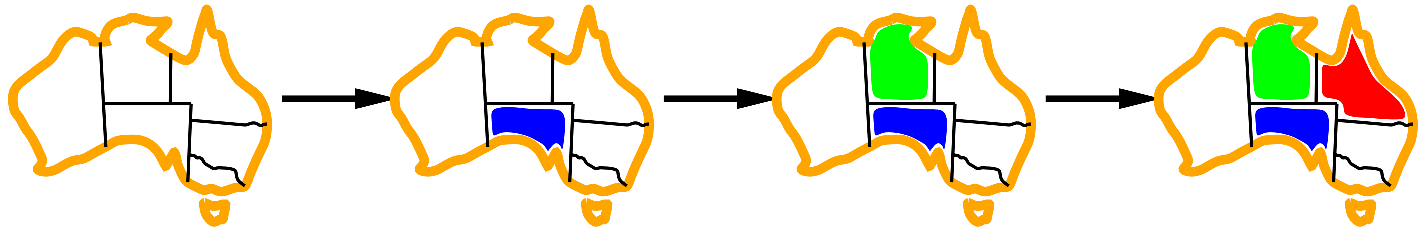


Degree Heuristic

Tie-breaker among MRV variables

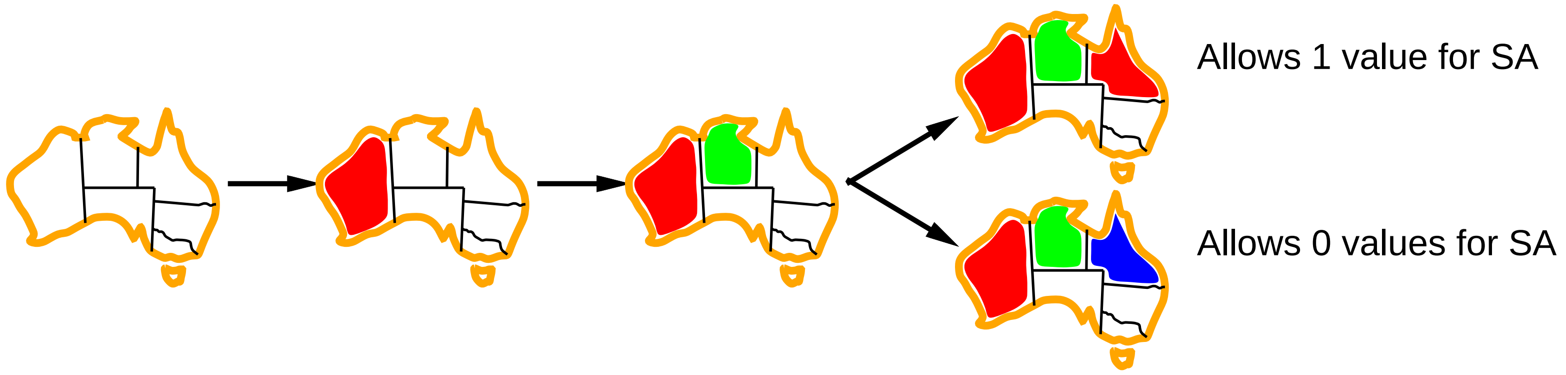
Degree heuristic:

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

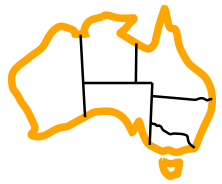


(More generally, 3 allowed values would be better than 2, etc.)

Combining these heuristics makes 1000 queens feasible.

Forward checking

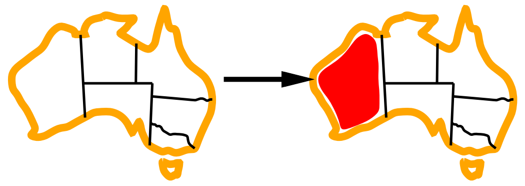
Idea: Keep track of remaining legal values for unassigned variables



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

Forward checking

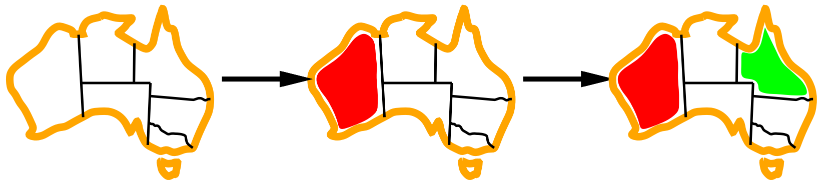
Idea: Keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values



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

Forward checking

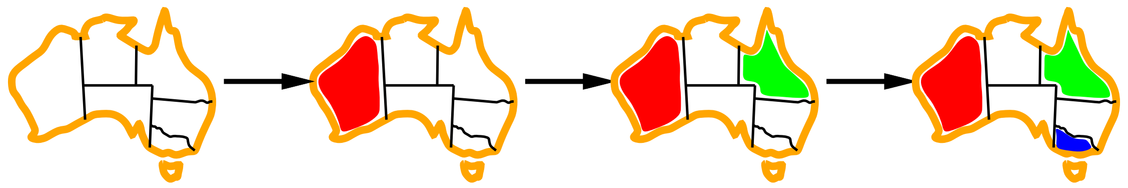
Idea: Keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values



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

Forward checking

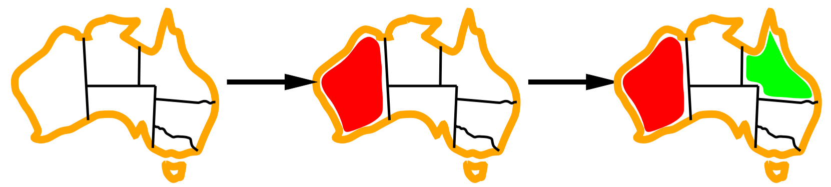
Idea: Keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values



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:



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

NT and SA cannot both be blue!

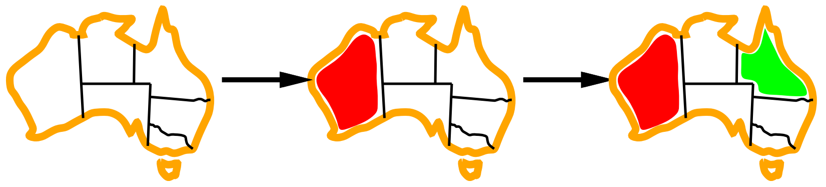
Constraint propagation repeatedly enforces constraints locally.

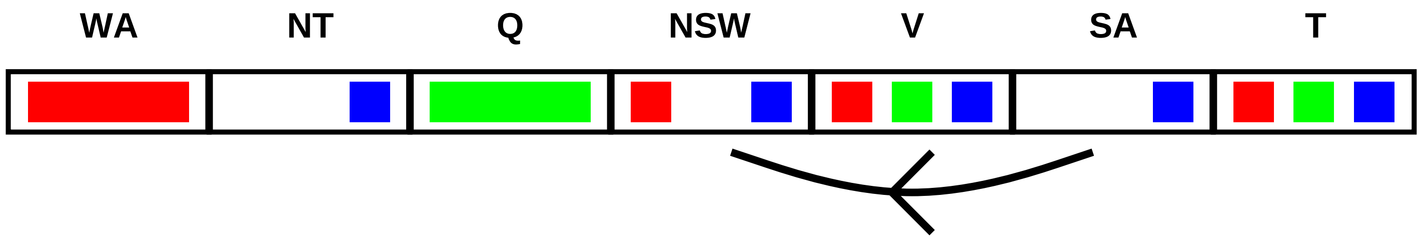
Arc consistency

Simplest form of constraint propagation makes each arc consistent

$X \rightarrow Y$ is consistent if

for every value x of X there is some allowed y



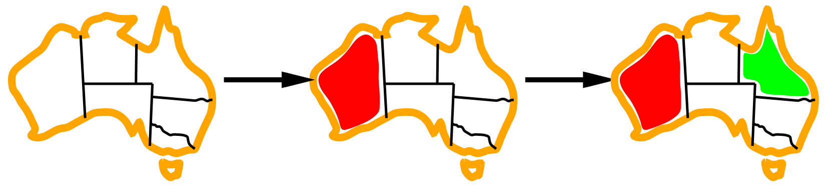


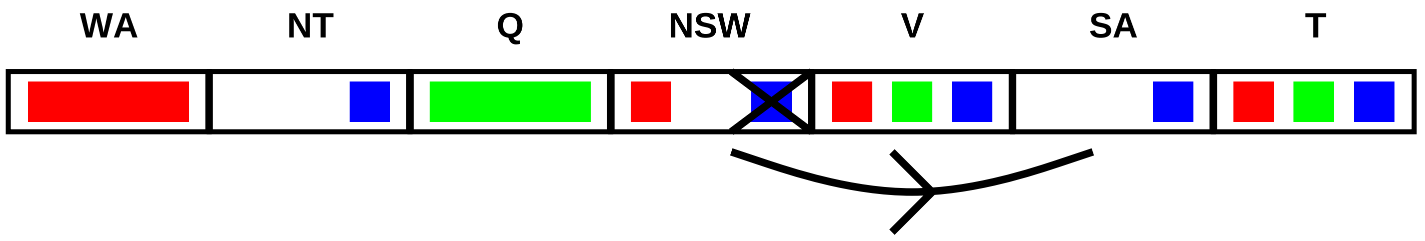
Arc consistency

Simplest form of propagation makes each arc consistent

$X \rightarrow Y$ is consistent if

for every value x of X there is some allowed y

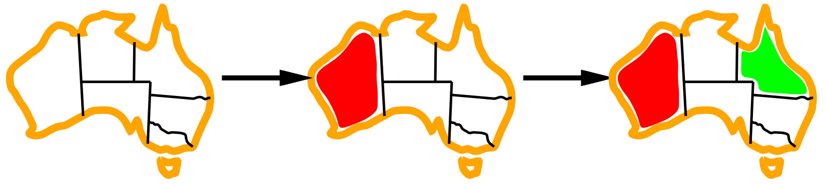


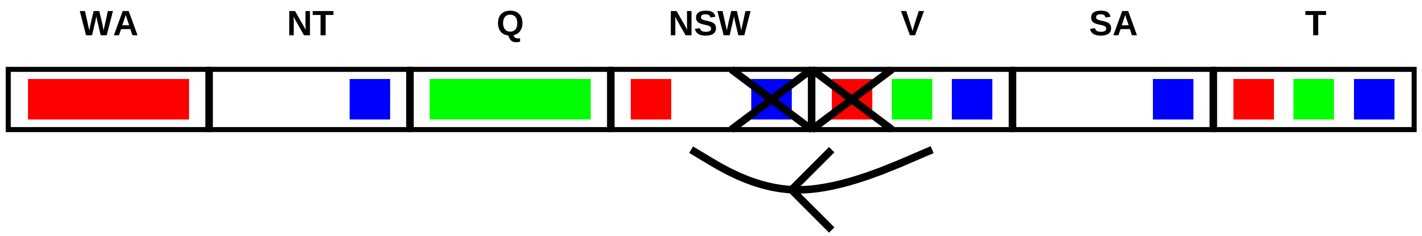


Arc consistency

$X \rightarrow Y$ is consistent if

for every value x of X there is some allowed y



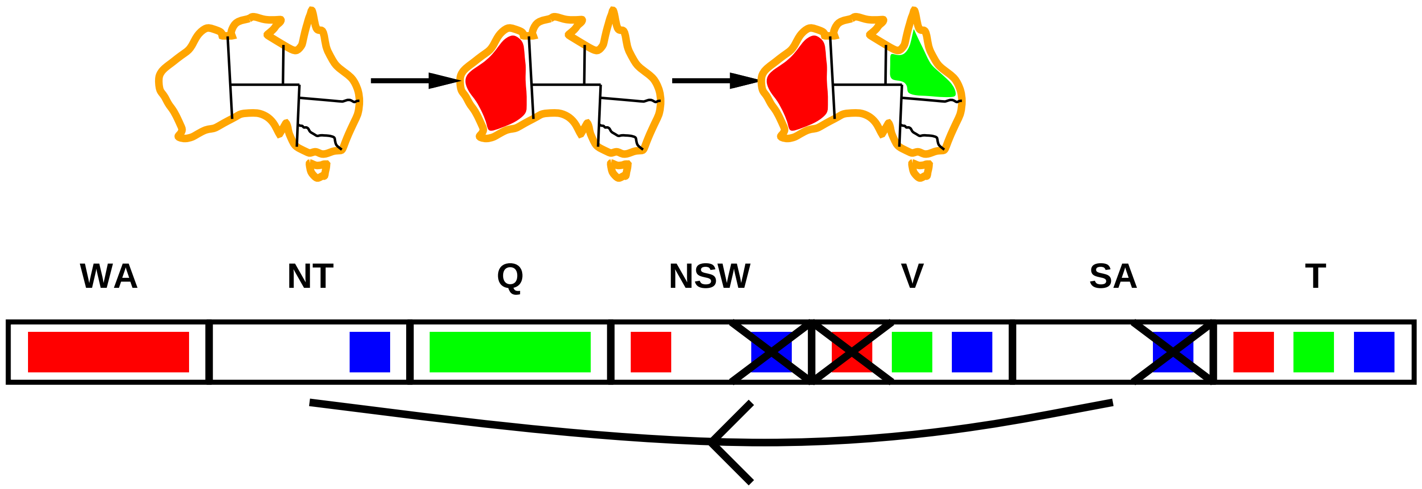


If X loses a value, neighbors of X need to be rechecked.

Arc consistency

$X \rightarrow Y$ is consistent if

for every value x of X there is some allowed y



Arc consistency detects failure earlier than forward checking.

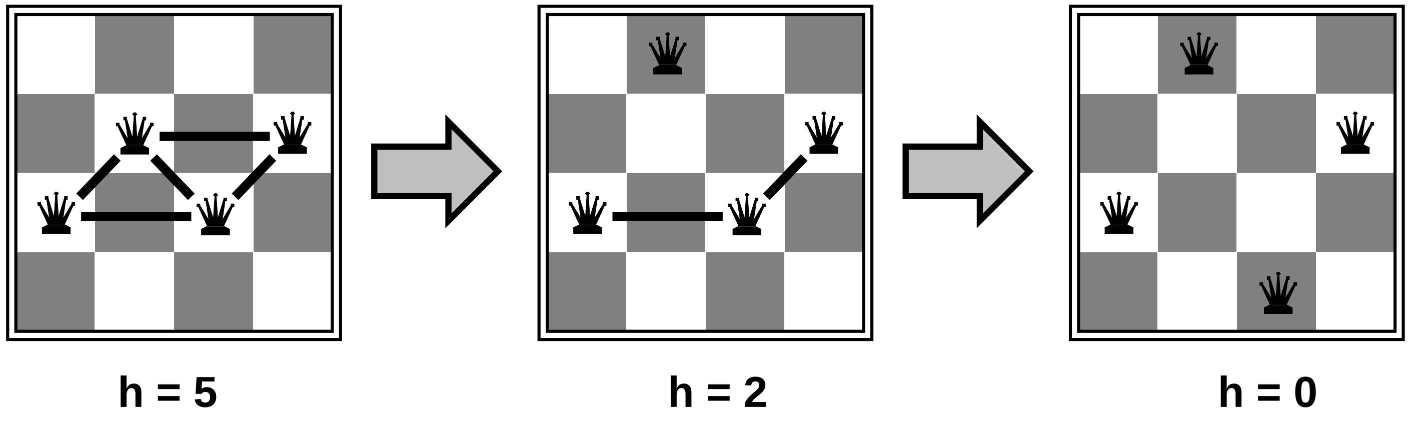
For some problems, it can speed up the search enormously.

For others, it may slow the search due to computational overheads.

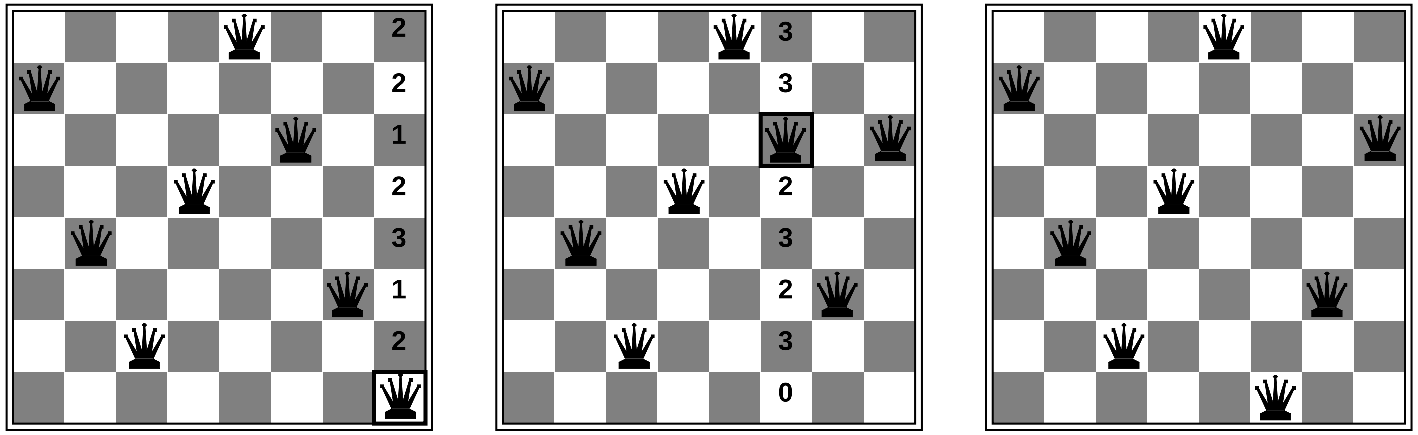
Local Search

There is another class of algorithms for solving CSP's, called "Iterative Improvement" or "Local Search".

These algorithms assign all variables randomly in the beginning (thus violating several constraints), and then change one variable at a time, trying to reduce the number of violations at each step.



Hill-climbing by min-conflicts



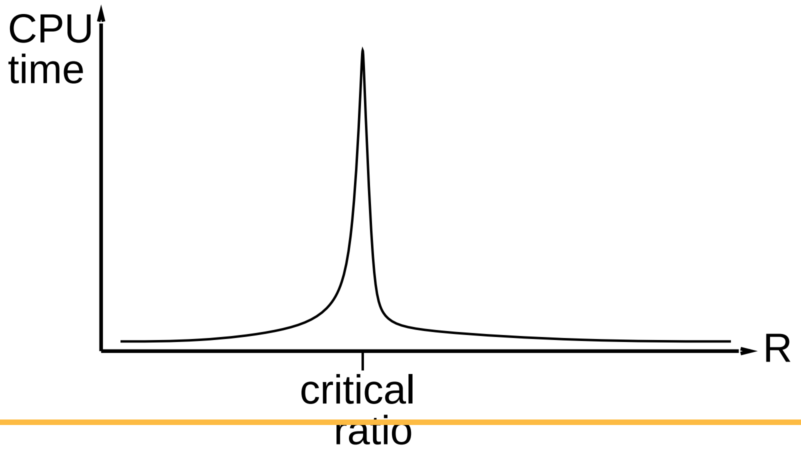
- Variable selection: randomly select any conflicted variable
- Value selection by min-conflicts heuristic
  - choose value that violates the fewest constraints

Phase transition in CSP's

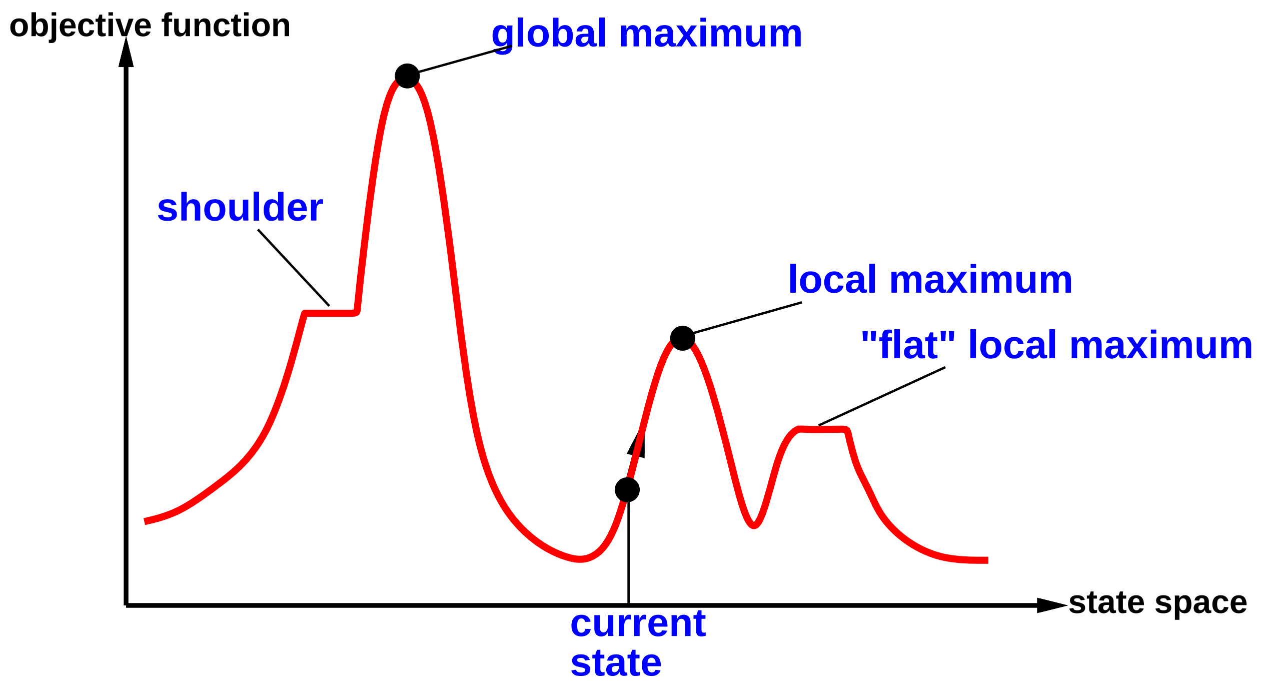
Given random initial state, hill climbing by min-conflicts can solve n-queens in almost constant time for arbitrary n with high probability (e.g., n = 10,000,000).

In general, randomly-generated CSP's tend to be easy if there are very few or very many constraints. They become extra hard in a narrow range of the ratio

$$R = \frac{\text{number of constraints}}{\text{number of variables}}$$



Flat regions and local optima



Sometimes, have to go sideways or even backwards in order to make progress towards the actual solution.

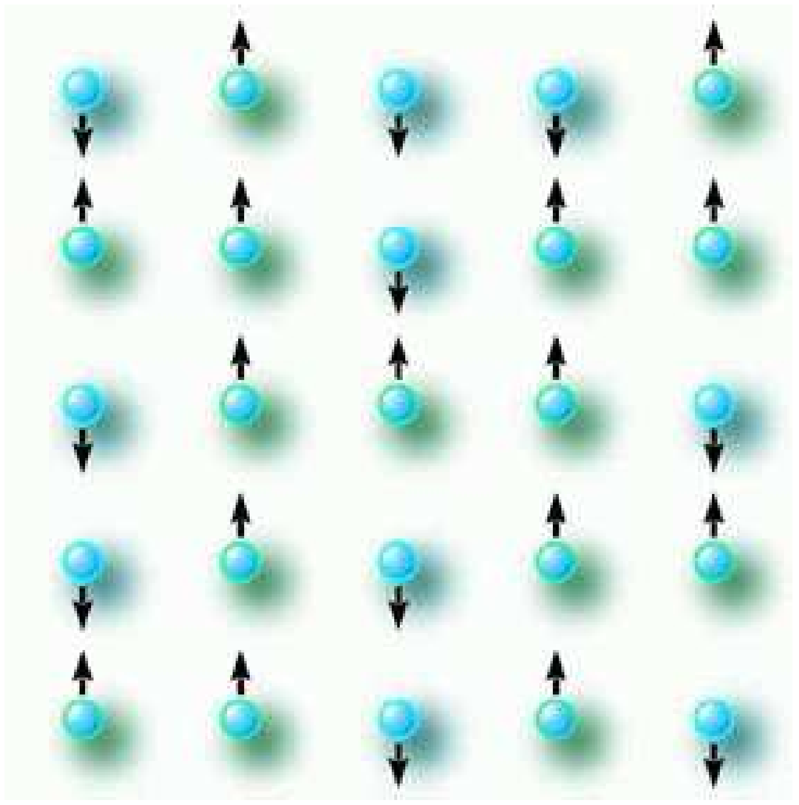
Simulated Annealing

- stochastic hill climbing based on difference between evaluation of previous state (h_0) and new state (h_1).
- if $h_1 < h_0$, definitely make the change
- otherwise, make the change with probability

  $$e^{-(h_1-h_0)/T}$$

  where T is a “temperature” parameter.
- reduces to ordinary hill climbing when $T = 0$
- becomes totally random search as $T \to \infty$
- sometimes, we gradually decrease the value of T during the search

Ising Model of Ferromagnetism



Summary

- Much interest in CSP’s for real-world applications
- Backtracking = depth-first search with one variable assigned per node
- Variable and Value ordering heuristics help significantly
- Forward Checking helps by detecting inevitable failure early
- Hill Climbing by min-conflicts often effective in practice
- Simulated Annealing can help to escape from local optima
- Which method(s) are best? It varies from one task to another!