COMP9414: Artificial Intelligence

Lecture 3a: Constraint Satisfaction

Wayne Wobcke

e-mail:w.wobcke@unsw.edu.au

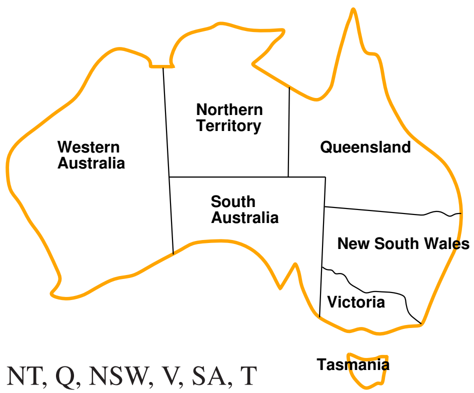
This Lecture

- Constraint Satisfaction Problems (CSPs)
- Standard search methods
  - Backtracking search and heuristics
  - Forward checking and arc consistency
  - Domain splitting and arc consistency
  - Variable elimination
- Local search
  - Hill climbing
  - Simulated annealing

Constraint Satisfaction Problems

- 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
- Aim is to find an assignment to each the variables X_i (a value from the domain D_i) such that all of the constraints C are satisfied

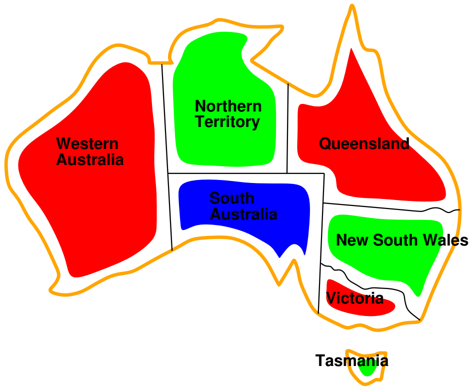
Example: Map Colouring



Variables: WA, NT, Q, NSW, V, SA, T
Domains: $D_i = \{\text{red, green, blue}\}$
Constraints: Adjacent regions have different colours (WA $\neq$ NT, etc.)

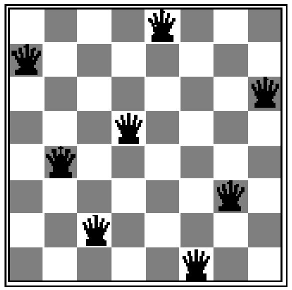
Example: Map Colouring

- Solution is an assignment that satisfies all the constraints



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

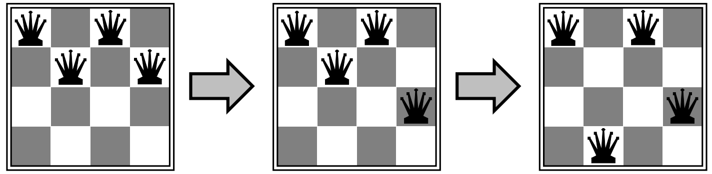
Example: n-Queens Puzzle



- Put n queens on $n \times n$ board so that no two queens attack one another

n-Queens Puzzle as a CSP

Assume one queen in each column. Domains are possible positions of queen in a column. Assignment is when each domain has one element.



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

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

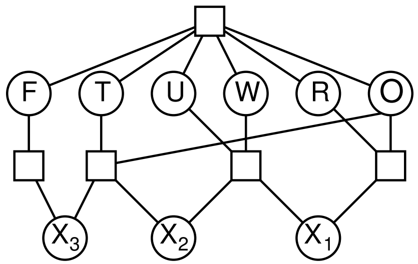
Variables:
D E M N O R S Y
Domains:
$\{0,1,2,3,4,5,6,7,8,9\}$

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 Hidden Variables

We can add "hidden" variables to simplify the constraints

```
   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:
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 (e.g. minimize space for circuit layout)
- Transport scheduling (e.g. courier delivery, vehicle routing)
- Factory scheduling (optimize assignment of jobs to machines)
- Gate assignment (assign gates to aircraft to minimize transit)

Many real world CSPs are also optimization problems

Varieties of CSPs

Discrete variables

- Finite domains; size $d \Rightarrow O(d^n)$ complete assignments
  - e.g. Boolean CSPs, incl. Boolean satisfiability (NP-complete)
- Infinite domains (integers, strings, etc.)
  - Job shop scheduling, variables are start/end days for each job
  - Need a constraint language, e.g. $\text{StartJob}_1 + 5 \leq \text{StartJob}_3$
  - 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

Types of Constraint

- 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!

Backtracking Search

CSPs can be solved by assigning values to variables one by one, in different combinations. Whenever a constraint is violated, go back to the most recently assigned variable and assign it a new value.

Can be implemented using Depth First Search on a special kind of state space, where states are defined by the values assigned so far

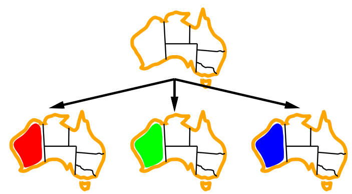
- Initial state: Empty assignment
- Successor function: Assign a value to an unassigned variable that does not conflict with previously assigned values of other variables
- Goal state: All variables are assigned a value and all constraints are satisfied

Path Search vs Constraint Satisfaction

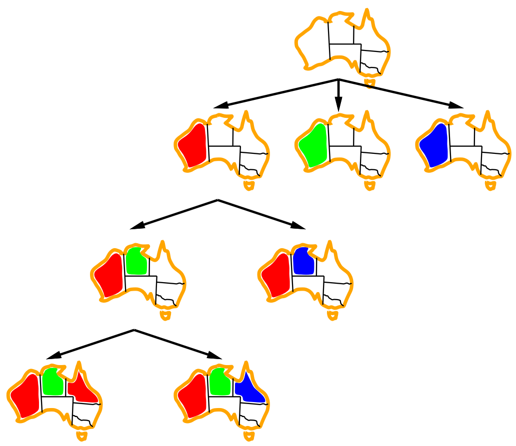
Important difference between path search problems and CSPs

- Constraint Satisfaction Problems (e.g. n-Queens)
  - Difficult part is knowing the final state
  - How to get there is easy
- Path Search Problems (e.g. Rubik's Cube)
  - Knowing the final state is easy
  - Difficult part is how to get there

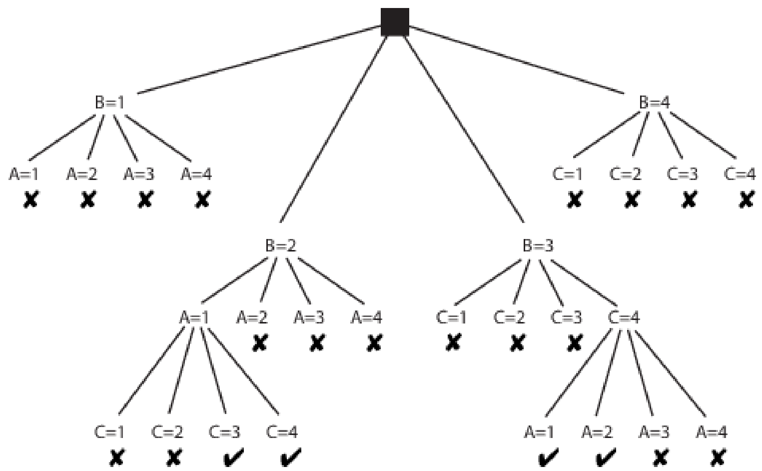
Backtracking Search Example



Backtracking Search Example



Problem with Backtracking Search



Backtracking Search Space Properties

The search space has very specific properties

- If there are n variables, every solution is at depth exactly n
- Variable assignments are commutative 交换的
  [WA = red then NT = green] same as [NT = green then WA = red]

Backtracking search can solve n-Queens for $n \approx 25$

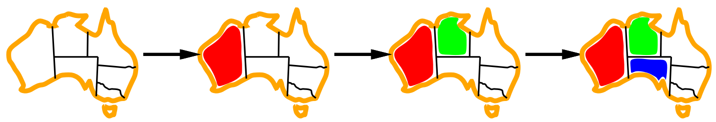
Improvements to Backtracking Search

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 inevitable failure be detected early?

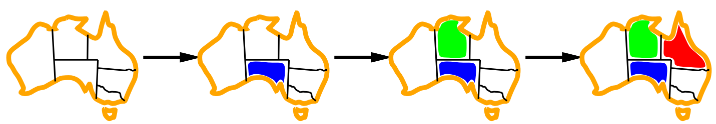
Minimum Remaining Values

- Minimum Remaining Values (MRV)
  - Choose the variable with the fewest legal values



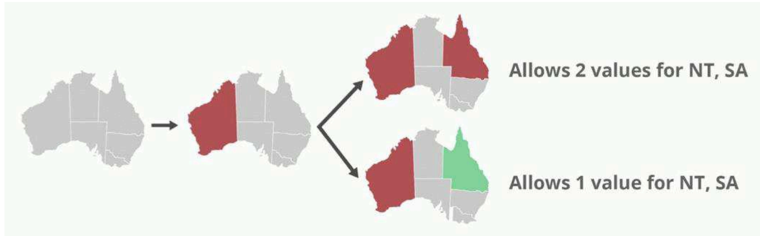
Degree Heuristic

- Tie-breaker among MRV variables
  - Choose variable with most constraints on remaining variables



Least Constraining Value

- Given a variable, choose the least constraining value
  - One that rules out the fewest values in the remaining variables



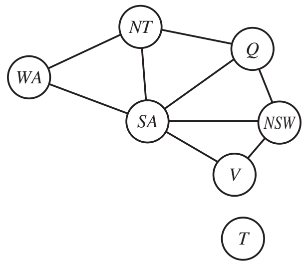
More generally, 3 allowed values would be better than 2, etc.

Combining these heuristics makes 1000-Queens feasible

Constraint Graph

Binary CSP: Each constraint relates at most two variables

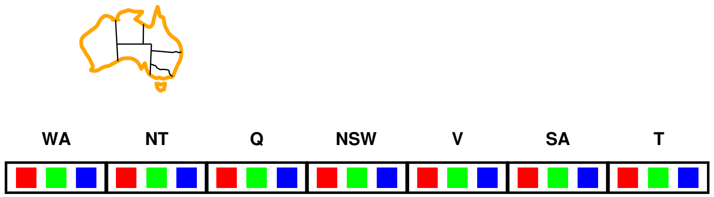
Constraint Graph: Nodes are variables, arcs show constraints



General-purpose CSP algorithms use the graph structure to speed up search, e.g. Tasmania is an independent subproblem!

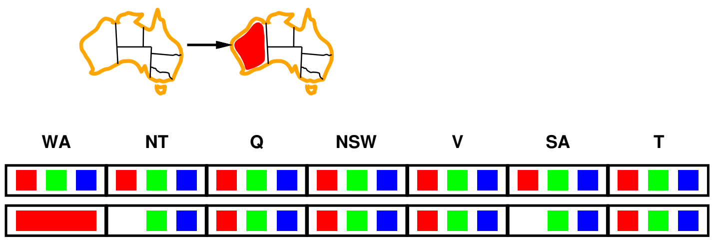
Forward Checking

Idea: Keep track of remaining legal values for unassigned variables



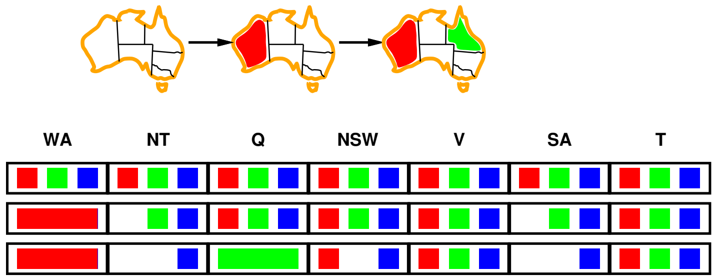
Forward Checking

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



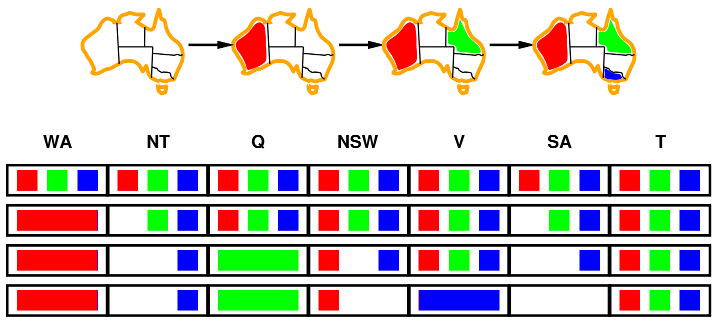
Forward Checking

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



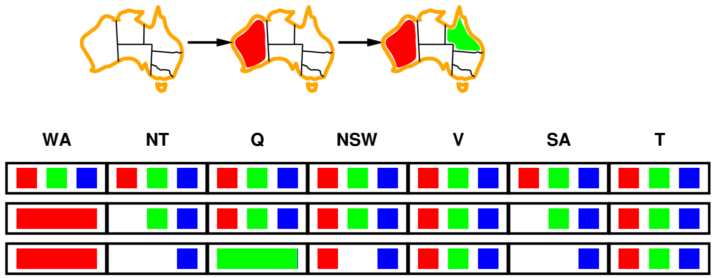
Forward Checking

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



Constraint Propagation

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!

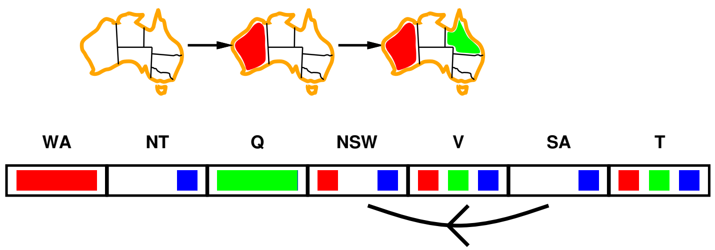
Constraint propagation repeatedly enforces constraints locally

Arc Consistency

Simplest form of constraint propagation makes each arc consistent

$X \to Y$ is consistent if

for every value x in $dom(X)$ there is some allowed y in $dom(Y)$



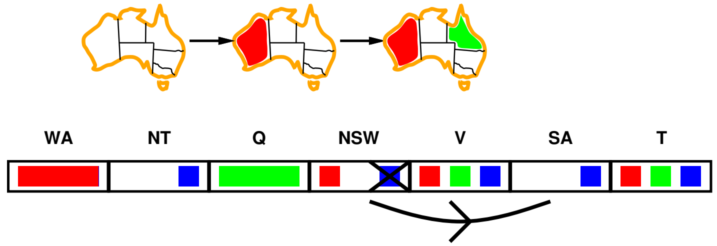
Make $X \to Y$ arc consistent by removing any such x from $dom(X)$

Arc Consistency

Simplest form of propagation makes every arc consistent

$X \to Y$ is consistent if

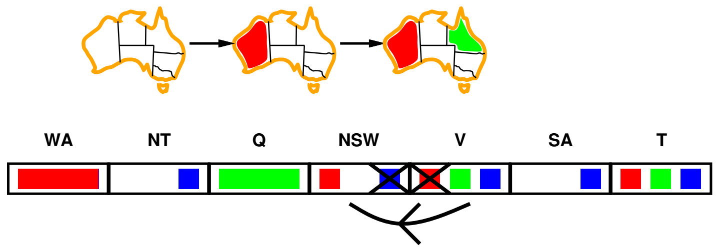
for every value x in $dom(X)$ there is some allowed y in $dom(Y)$



Arc Consistency

$X \to Y$ is consistent if

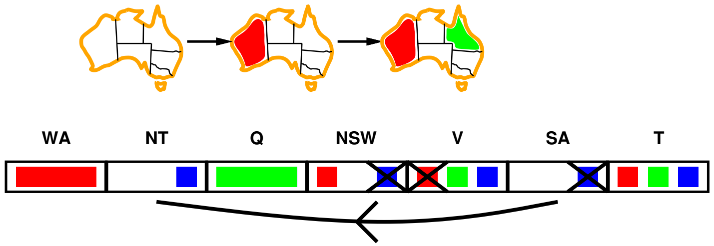
for every value x in $dom(X)$ there is some allowed y in $dom(Y)$



If X loses a value, neighbours of X need to be rechecked

Arc Consistency

$X \to Y$ is consistent if
for every value x in $dom(X)$ there is some allowed y in $dom(Y)$



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

Domain Splitting and Arc Consistency

States are whole CSPs (not partial assignments)

- Make CSP domain consistent and arc consistent
  - Domain consistent = all unary constraints are satisfied
- To solve CSP using Depth First Search
  - Choose a variable v with more than one value in domain
  - Split the domain of v into two subsets
  - This gives two smaller CSPs
  - Make each CSP arc consistent
  - Solve each resulting CSP (or backtrack if unsolvable)

Constraint Optimization Problems

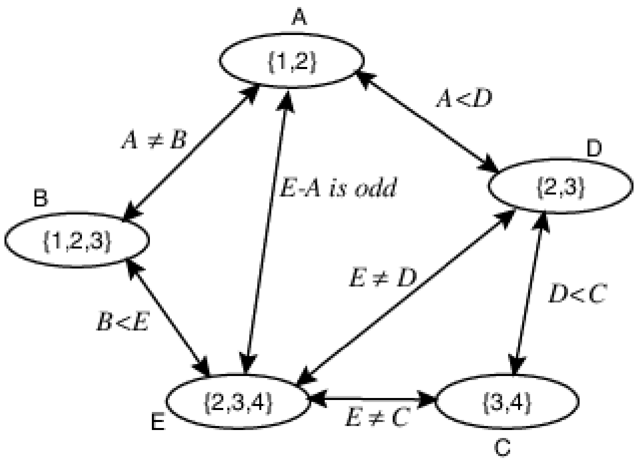
States are whole CSPs (not partial assignments) with costs

- Make CSP domain consistent and arc consistent
- Add CSP to priority queue
- To solve CSP using Greedy Search
  - Remove CSP with minimal h from priority queue
  - Choose a variable v with more than one value in domain
  - Split the domain of v into two subsets
  - This gives two smaller CSPs
  - Make each CSP arc consistent – add to priority queue
- $cost$(CSP) = sum of costs to violate soft constraints

Variable Elimination

- If there is only one variable, return the intersection of its (unary) constraints
- Otherwise
  - Select a variable X
  - Join the constraints in which X appears, forming constraint R1
  - Project R1 onto its variables other than X, forming R2
  - Replace all of the constraints in which X appears by R2
  - Recursively solve the simplified problem, forming R3
  - Return R1 joined with R3

Variable Elimination Example



Variable Elimination Example

$r_1 : C \neq E$	C	E
	3	2
	3	4
	4	2
	4	3

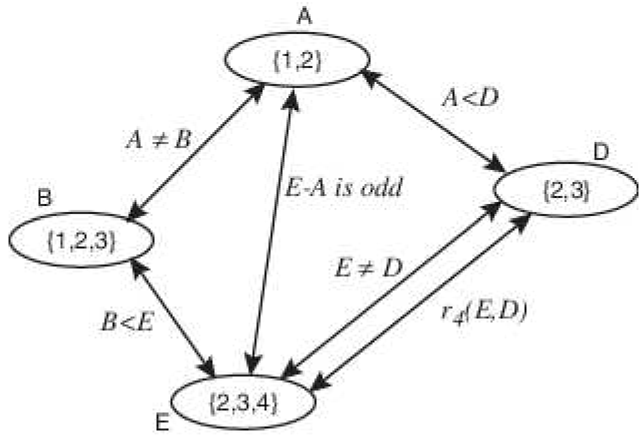
$r_2 : C > D$	C	D
	3	2
	4	2
	4	3

$r_3 : r_1 \bowtie r2$	C	D	E
	3	2	2
	3	2	4
	4	2	2
	4	2	3
	4	3	2
	4	3	3

$r_4 : \pi_{\{D,E\}} r_3$	D	E
	2	2
	2	3
	2	4
	3	2
	3	3

$\hookrightarrow$ new constraint

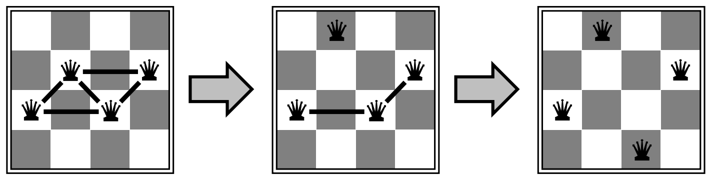
Variable Elimination Example



not guaranteed to find a solution..

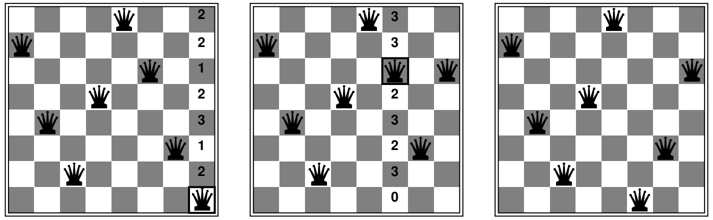
Local Search

- Iterative Improvement
  - Assign all variables randomly (possibly violating constraints)
  - Change one variable at a time, trying to reduce the number of violations at each step
  - Greedy Search with h = number of constraints violated



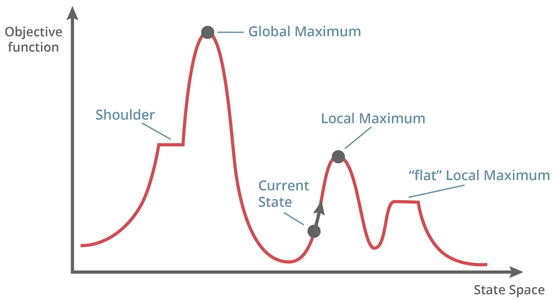
h = 5 h = 2 h = 0

Hill Climbing by Min-Conflicts



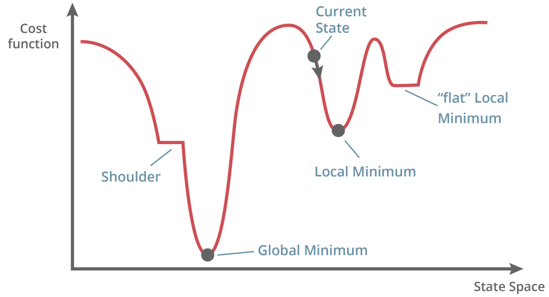
- Variable selection: randomly select any conflicted variable
- Value selection by min-conflicts heuristic
  - Choose value that violates fewest constraints
  - Can (often) solve n-Queens for $n \approx 10,000,000$

Plateaux and Local Optima



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

Inverted View



When minimizing violated constraints, it makes sense to think of starting at the top of a ridge and climbing down into the valleys

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

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

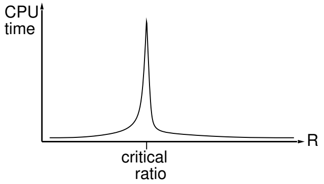
  where T is a "temperature" parameter
- Reduces to ordinary hill climbing when $T = 0$
- Becomes totally random search as $T \to \infty$
- Sometimes, gradually decrease value of T during search

Phase Transitions in CSPs

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

Randomly-generated CSPs tend to be easy if there are very few or very many constraints, but become extra hard in a narrow range of the ratio

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



Summary

- Much interest in CSPs 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 varies from one task to another!