

Constraint Satisfaction Problems

COMP3411/9814: Artificial Intelligence

Constraint Satisfaction Problems

- Assignment problems (e.g. who teaches what class)
- Timetabling problems (e.g. which class is offered when and where?)
- Hardware configuration (e.g. minimise space for circuit layout)
- Transport scheduling (e.g. courier delivery, vehicle routing)
- Factory scheduling (optimise assignment of jobs to machines)
- Gate assignment (assign gates to aircraft to minimise transit)

Closely related to optimisation problems

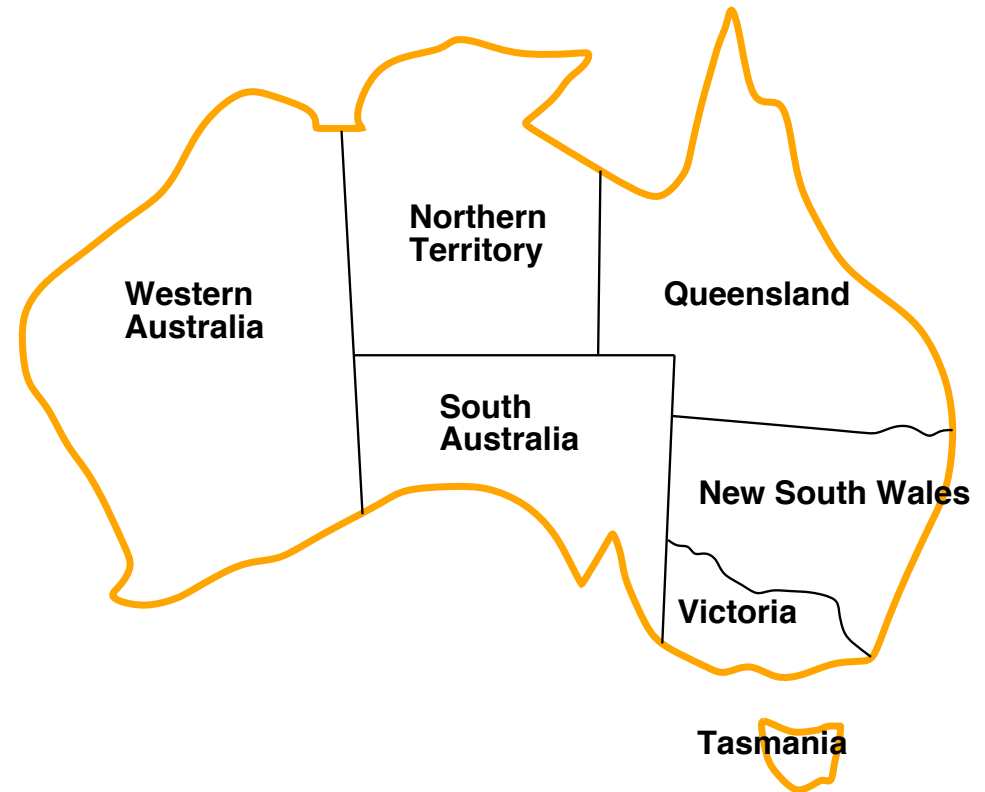
Lecture Overview

- Constraint Satisfaction Problems (CSPs)
- CSP examples
- 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 (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 that specify allowable combinations of values.
- 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-Colouring



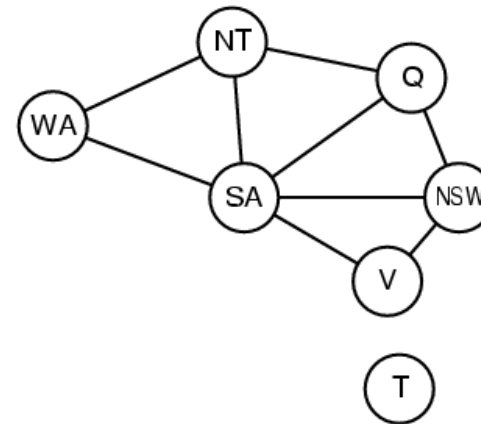
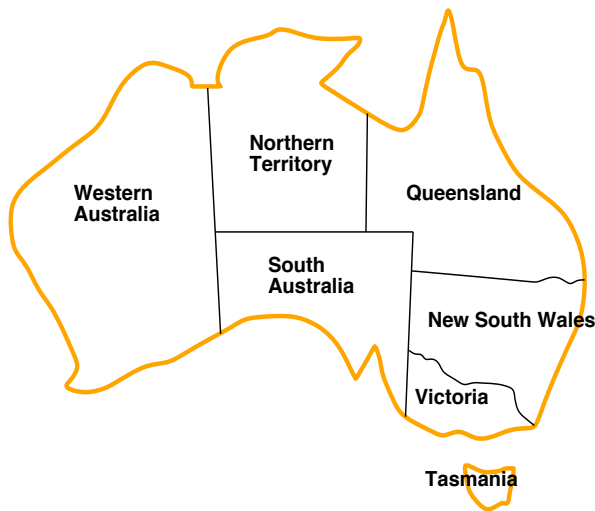
Variables: WA, NT, Q, NSW, V, SA, T

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

Constraints: adjacent regions must have different colours
e.g. $WA \neq NT$, etc.

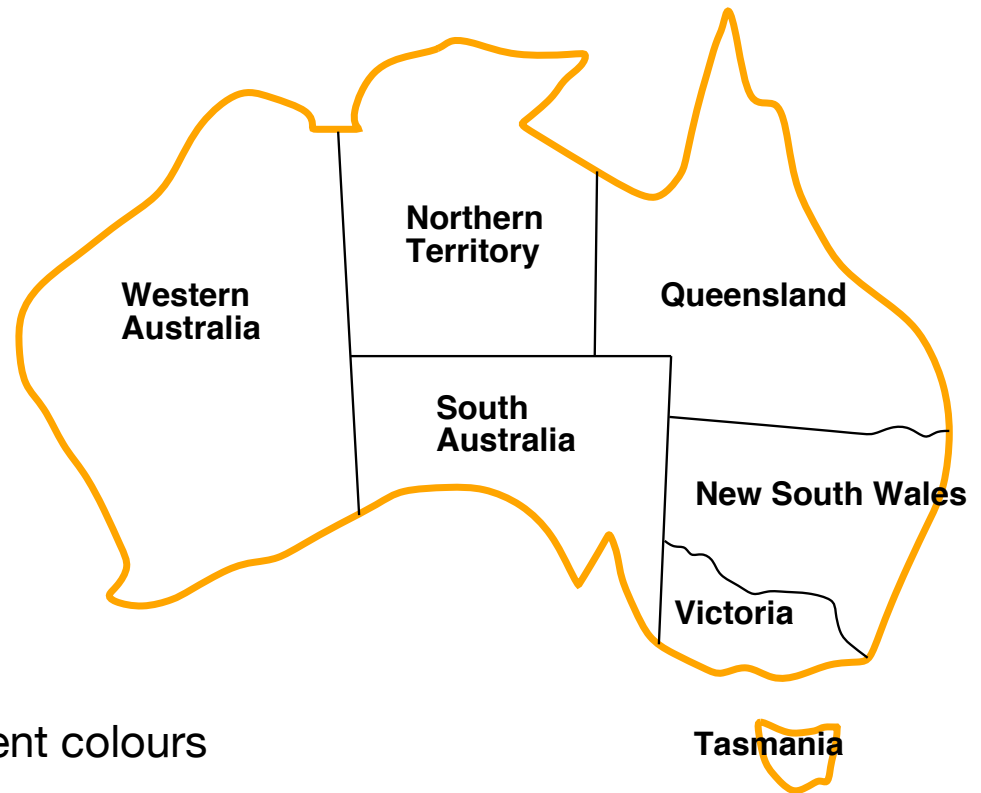
Constraint graph

Constraint graph: nodes are variables, arcs are constraints



Binary CSP: each constraint relates two variables

Example: Map-Colouring



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

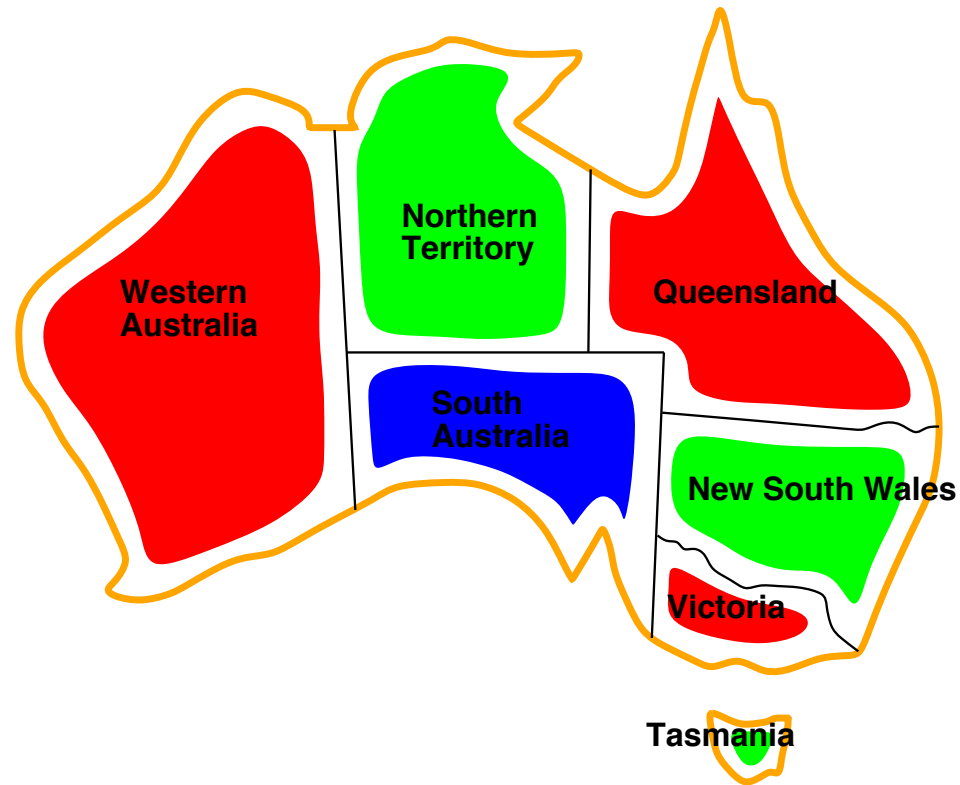
Domains: $D_i = \{\text{red, green, blue}\}$

Constraints: adjacent regions must have different colours

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

or (WA,NT) in $\{(\text{red,green}),(\text{red,blue}),(\text{green,red}), (\text{green,blue}),(\text{blue,red}),(\text{blue,green})\}$

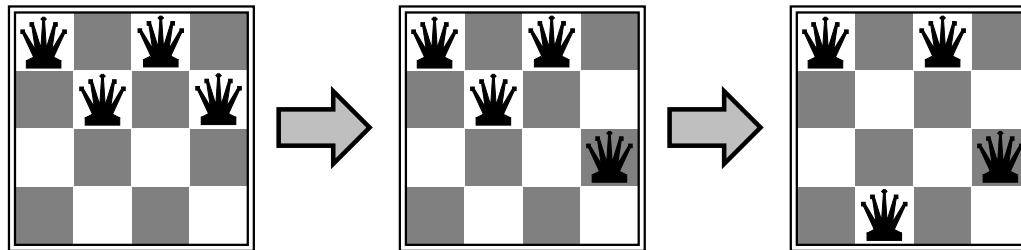
Example: Map-Colouring



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

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. Which row does each one go in?



Variables: Q_1, Q_2, Q_3, Q_4

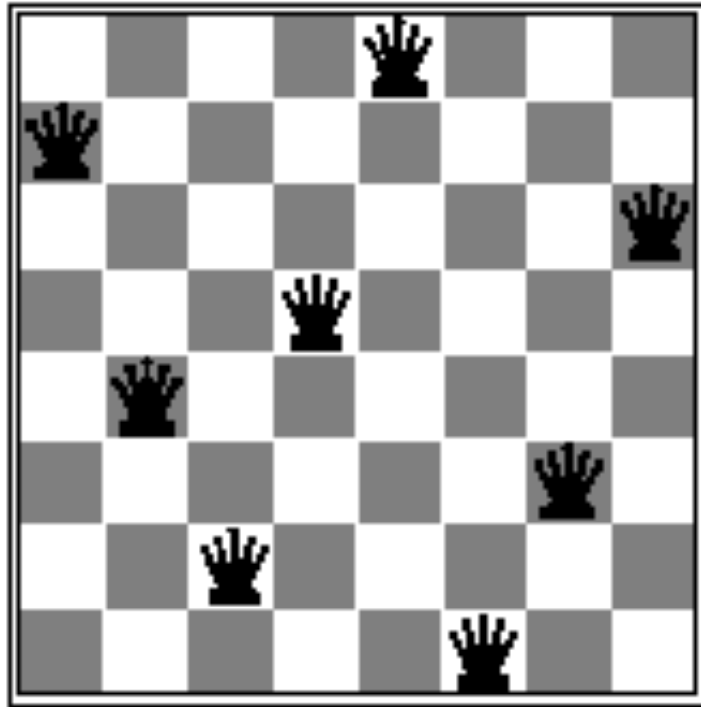
Domains: $D_i = \{1, 2, 3, 4\}$

Constraints:

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

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

Example: n-Queens Puzzle



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

Example: Cryptarithmic

$$\begin{array}{r} \\ \\ + \\ \hline M \end{array}$$

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.

Example: Cryptarithmic

$$\begin{array}{r} \text{T W O} \\ + \text{T W O} \\ \hline \text{F O U R} \end{array}$$

Variables: F T U W R O X_1 X_2 X_3

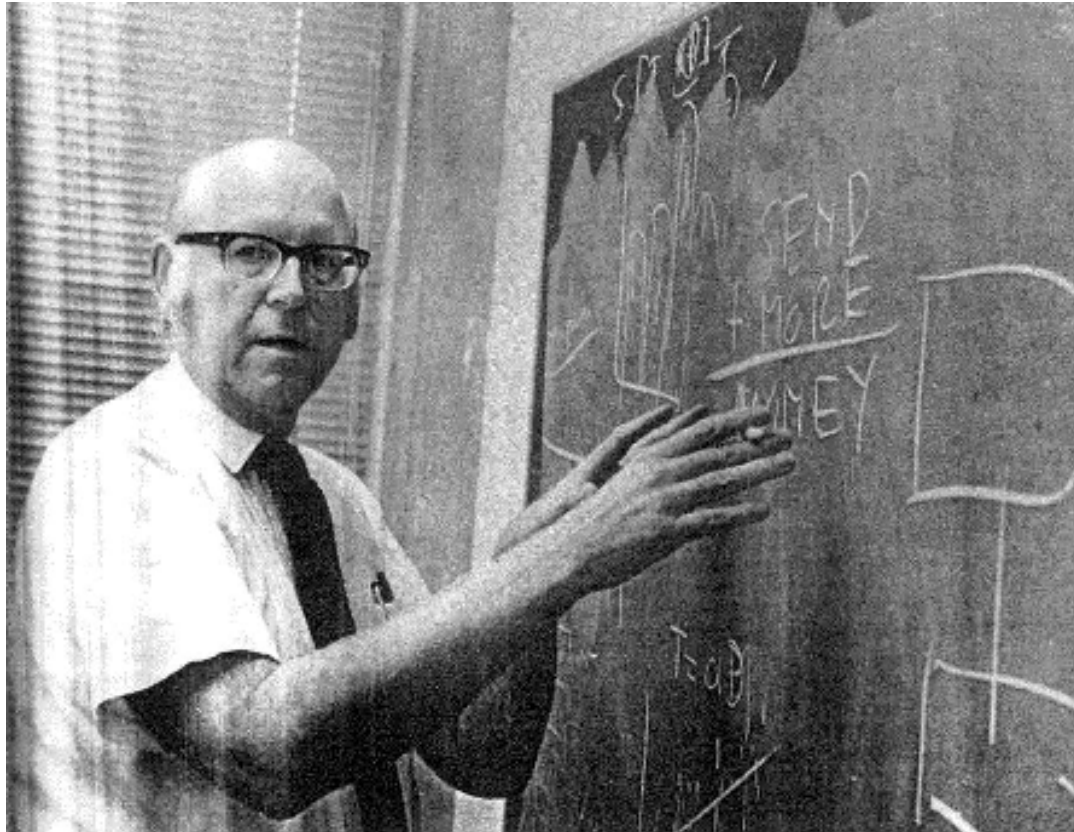
Domains: {0,1,2,3,4,5,6,7,8,9}

Constraints:

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

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

Cryptarithmic with Allen Newell



Book: Intended Rational Behavior

Cryptarithmic with Allen Newell

7.1. Cryptarithmic

Intendedly Rational Behavior ■ 365

Let us start with **cryptarithmic**. This task was first analyzed by Bartlett (1958) and later by Herb Simon and myself (Newell & Simon, 1972). It plays an important role in the emergence of cognitive psychology—at least for me, and perhaps for others. It has been the strongest convincer that humans really do use problem spaces and do search in them, just as the AI theory of heuristic search says.

A **cryptarithmic** task is just a small arithmetical puzzle (see Figure 7-1). The words *DONALD*, *GERALD*, and *ROBERT* represent three six-digit numbers. Each letter is to be replaced by a distinct digit (that is, *D* and *T* must each be a digit, say $D = 5$ and $T = 0$, but they cannot be the same digit). This replacement must lead to a correct sum, such that $DONALD + GERALD = ROBERT$. Mathematically viewed, the problem is one of satisfying multiple integer constraints involving equality, inequality, and inequality.

Humans can be set to solving **cryptarithmic** tasks, and pro-

Assign each letter a unique digit to make a correct sum

$$\begin{array}{r} DONALD \\ + GERALD \\ \hline ROBERT \end{array} \quad D = 5$$

Figure 7-1. The **cryptarithmic** task.

ocols can be obtained from transcripts of their verbalizations while they work (Newell & Simon, 1972). Analysis shows that people solve the task by searching in a problem space and that the search can be plotted explicitly.¹ Figure 7-2 shows the behavior of S3 on $DONALD + GERALD = ROBERT$.² The plot is called a *problem-behavior graph (PBG)*; it is just a way of spreading out the search so it can be examined (Figure 1-4 showed a PBG for chess). S3

Cryptarithmic with Allen Newell

366 ■ Unified Theories of Cognition

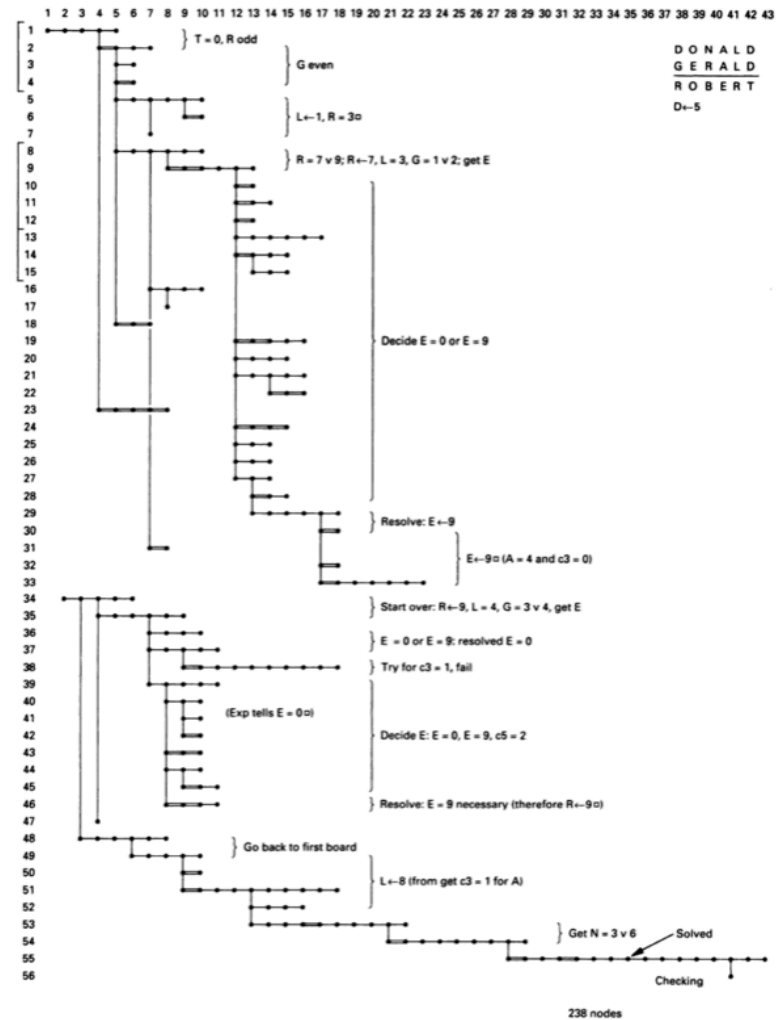
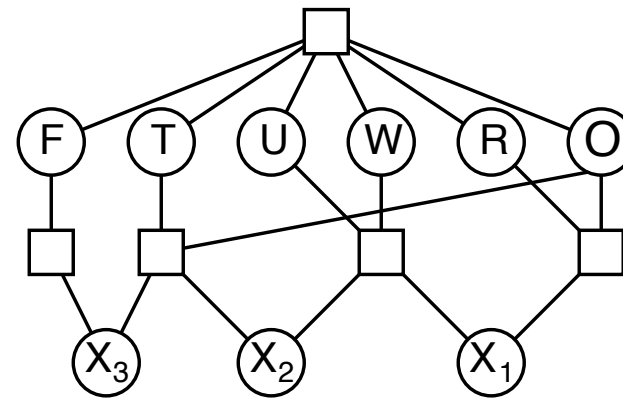


Figure 7-2. Problem-behavior graph of subject S3 on
 $DONALD + GERALD = ROBERT$.

Cryptarithmic with Auxiliary Variables

- We can add “auxiliary” variables to simplify the constraints, e.g. to help handle carry-over

$$\begin{array}{r} \text{ T W O} \\ + \text{ T W O} \\ \hline \text{ F O U R} \end{array}$$



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.

CSP Application - Factory scheduling

- A robot (agent) needs to schedule a set of activities for a manufacturing process, involving casting, milling, drilling, and bolting.
- Each activity has a set of possible times at which it may start.
- The robot has to satisfy various constraints arising from prerequisite requirements and resource use limitations.
- For each activity there is a variable that represents the time that it starts:
 - B – start of bolting
 - D – start of drilling
 - C – start of casting

CSP Application - Factory scheduling

Constraints on the possible dates for three activities:

Variables: A , B , C - variables that represent the date of each activity

Domain of each variable is: $\{1, 2, 3, \}$

A *constraint* with scope: $(A \leq B) \wedge (B < 3) \wedge (B < C) \wedge \neg(A = B \wedge C \leq 3)$

A starts on or before the same date as B and it cannot be that A and B start on the same date and C starts on or before day 3.

CSP Application - Factory scheduling

Constraint on the possible dates for three activities.

Variables: A, B, C - variables that represent the date of each activity

Domain of each variable is: $\{1, 2, 3, \}$

A *constraint* with scope:

$$(A \leq B) \wedge (B < 3) \wedge (B < C) \wedge \neg(A = B \wedge C \leq 3)$$

A starts on or before the same date as B and it cannot be that A and B start on the same date and C starts on or before day 3.

Constraint defines its **extension**, e.g. table specifying the legal assignments:

A	B	C
2	2	4
1	1	4
1	2	3
1	2	4

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 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
 - $\text{EndJob1} + 5 \leq \text{StartJob3}$
- **Soft** constraints (Preferences)
 - 11am lecture is better than 8am lecture!

Path Search vs Constraint Satisfaction

Difference between path search problems and CSPs

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

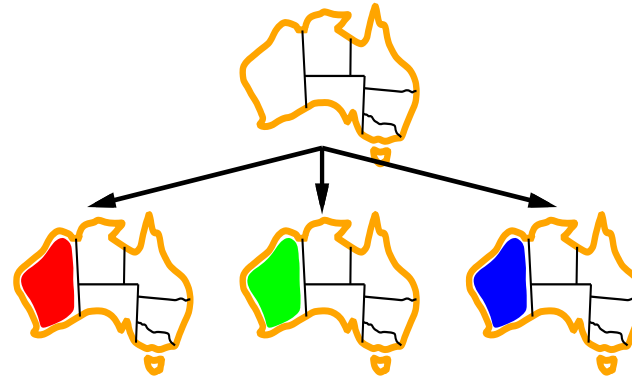
Backtracking Search

- CSPs can be solved by assigning values to variables one by one, in different combinations.
- Whenever a constraint is violated, go back to most recently assigned variable and assign it a new value.
- Can use Depth First Search, 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. (If no legal values remain, the successor function fails.)
 - Goal test: all variables have been assigned a value, and no constraints have been violated.

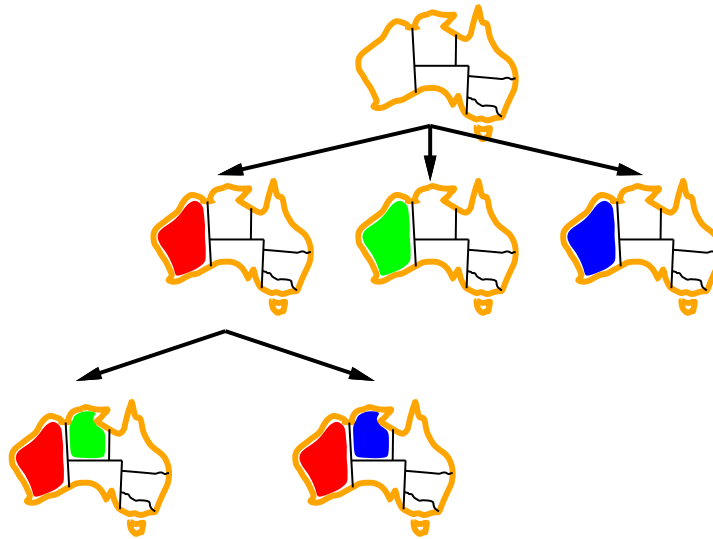
Backtracking example



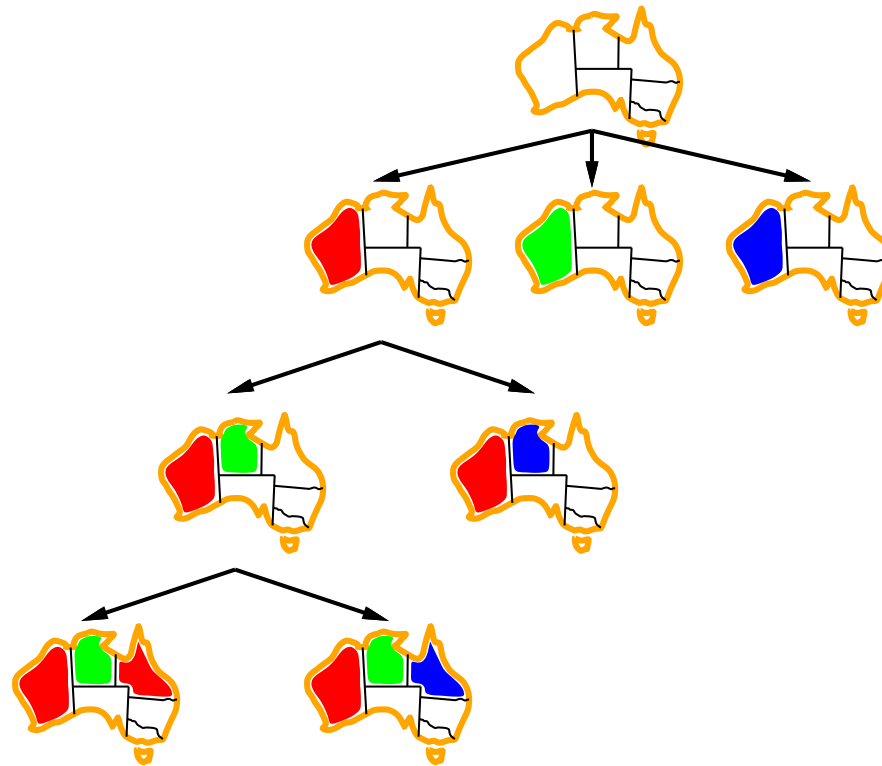
Backtracking example



Backtracking example



Backtracking example



Backtracking Search Properties

- If there are n variables, every solution will occur at exactly depth n .
- Variable assignments are commutative

[WA = red then NT = green] same as [NT = green then WA = red]

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

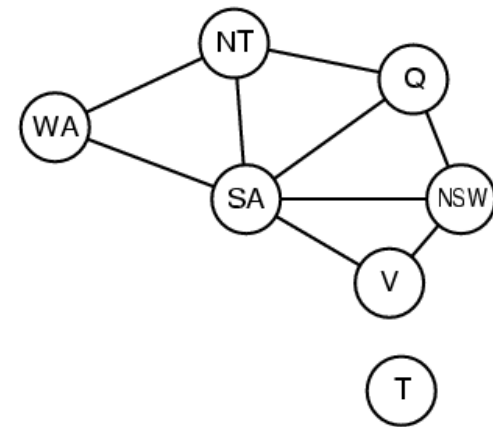
Programming Constraints

```
variables([wa=_, nt=_, q=_, nsw=_, v=_, sa=_, t=_]).
```

```
domain(red).  
domain(green).  
domain(blue).
```

```
connected(wa, nt).  
connected(wa, sa).  
connected(nt, q).  
connected(nt, sa).  
connected(sa, q).  
connected(sa, nsw).  
connected(sa, v).  
connected(q, nsw).  
connected(v, nsw).
```

```
adjacent(A, B) :- connected(A, B).  
adjacent(A, B) :- connected(B, A).
```



Programming Constraints

```
solve(V) :-  
    variables(V),  
    assign_all(V).
```

```
assign_all([]).  
assign_all([State|OtherVariables]):-  
    assign_all(OtherVariables),  
    assign_variable(State, OtherVariables).
```

```
assign_variable(Var = Colour, OtherVariables) :-  
    domain(Colour),  
    constraint(Var = Colour, OtherVariables).
```

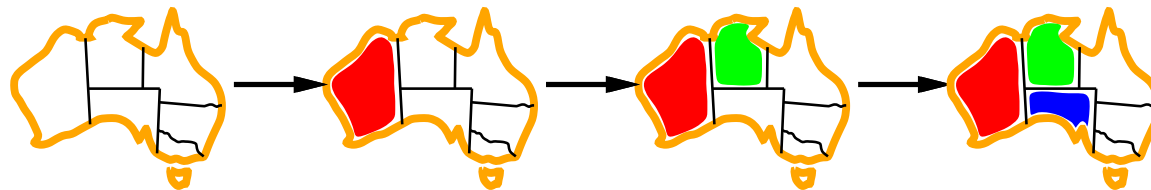
```
constraint(S1 = C, OtherVariables) :-  
    findall(S, (adjacent(S1, S), member(S = C, OtherVariables)), []).
```

Improvements to Backtracking Search

- Which variable should be assigned next?
- In what order should its values be tried?
- Can we detect inevitable failure early?

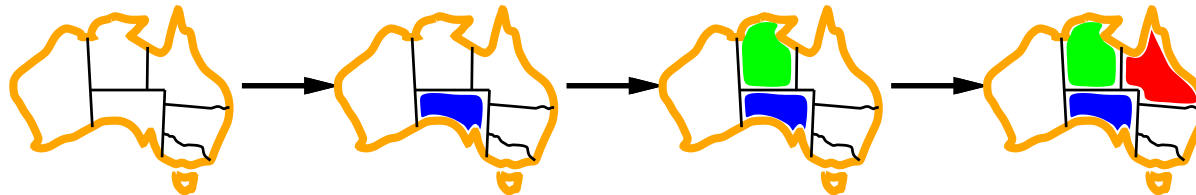
Minimum Remaining Values

- Minimum Remaining Values (MRV)
- choose the variable with the fewest legal values
- Most constrained variable



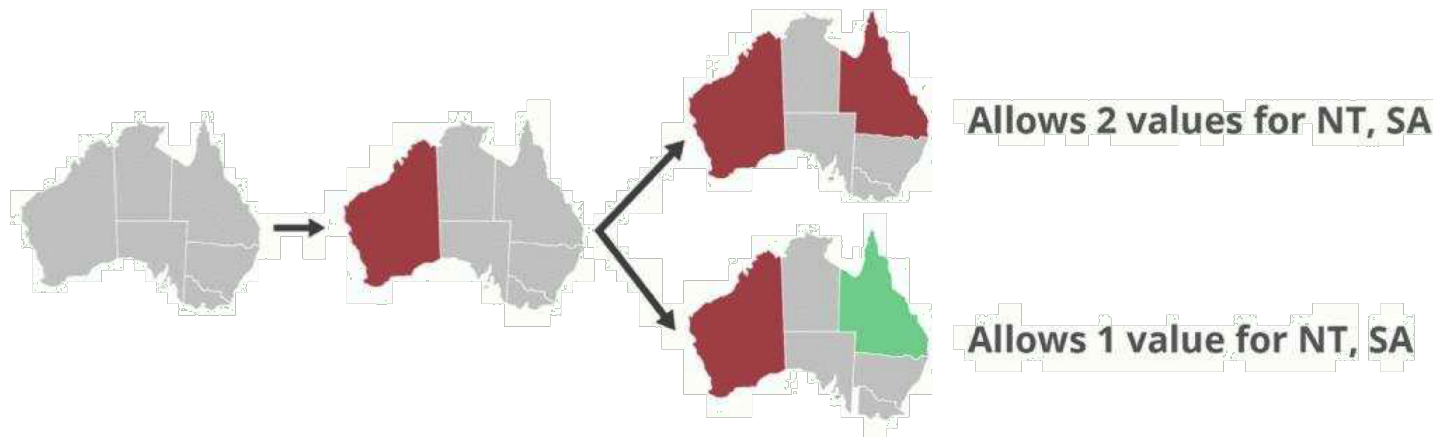
Degree Heuristic

- Tie-breaker among MVR 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

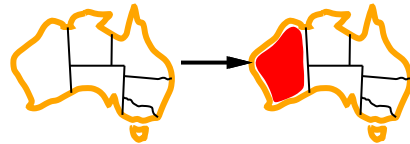
- Keep track of remaining legal values for unassigned variables
- Terminate search when any variable has no legal values
 - prune off that part of the search tree, and backtrack



Initially, all values are available.

Forward Checking

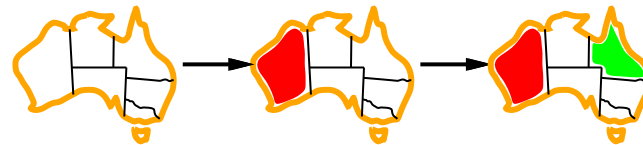
- Keep track of remaining legal values for unassigned variables
- Terminate search when any variable has no legal values









WA	NT	Q	NSW	V	SA	T
  	  	  	  	  	  	  
	 	  	  	  	 	  

Forward Checking

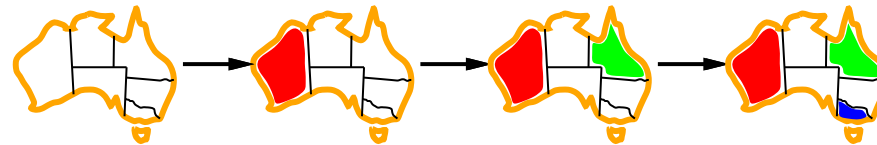
- Keep track of remaining legal values for unassigned variables
- Terminate search when any variable has no legal values






















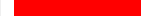







WA	NT	Q	NSW	V	SA	T
						
						
						

Forward Checking

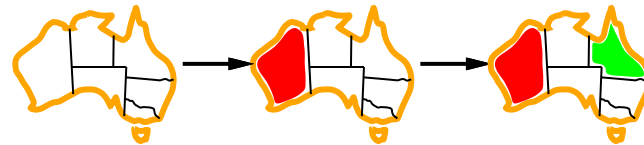
- Keep track of remaining legal values for unassigned variables
- Terminate search when any variable has no legal values



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
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>

NT and SA cannot both be blue!

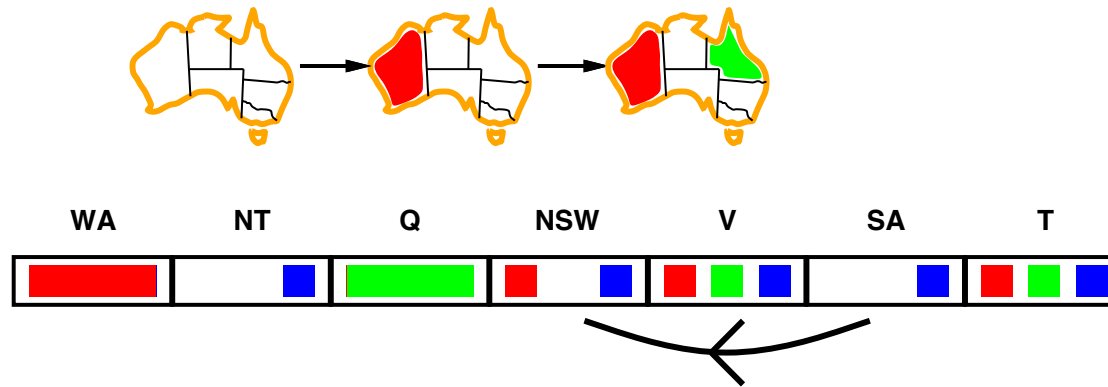
Constraint propagation repeatedly enforces constraints locally

Arc Consistency

Simplest form of propagation makes each arc consistent

$X \rightarrow Y$ is consistent if

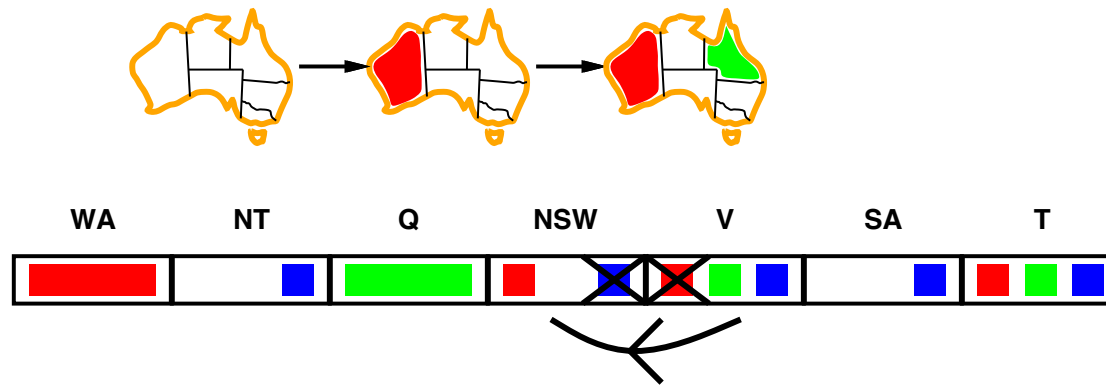
for **every** value x of X there is **some** allowed y



Arc Consistency

$X \rightarrow Y$ is consistent if

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

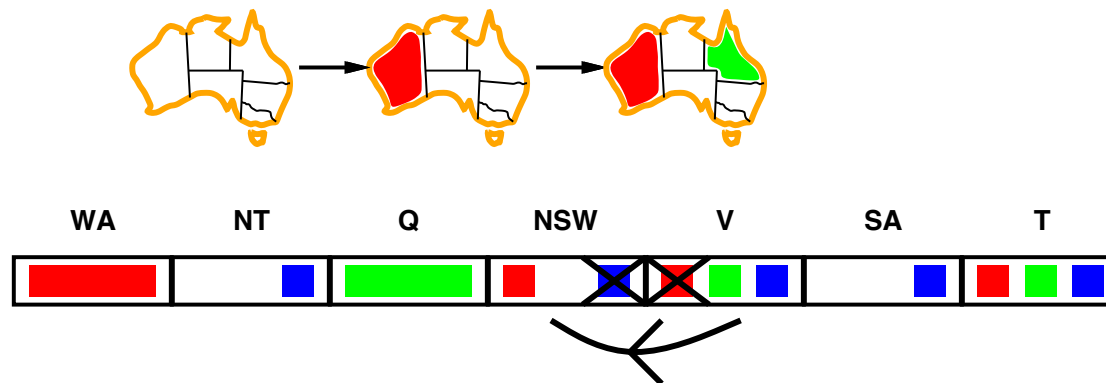


If X loses a value, neighbours of X need to be rechecked.

Arc Consistency

$X \rightarrow Y$ is consistent if

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

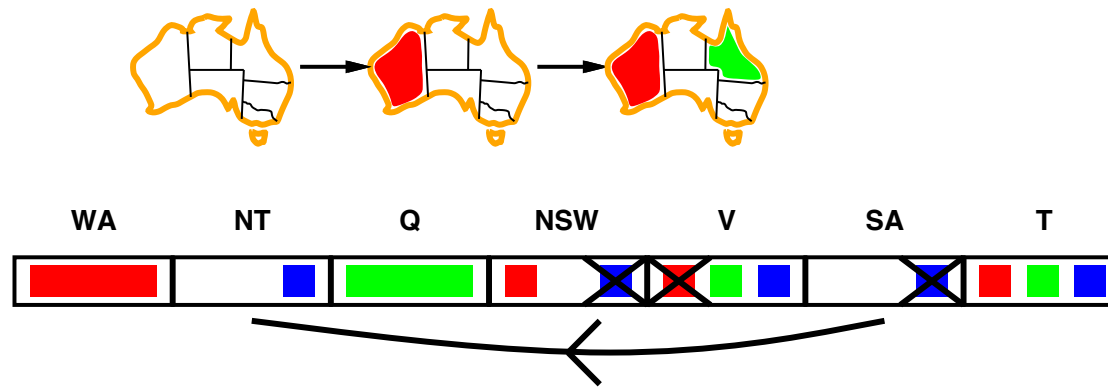


- If X loses a value, neighbours of X need to be rechecked.
- Arc consistency detects failure earlier than forward checking
- Can be run as a preprocessor after each assignment

Arc Consistency

$X \rightarrow Y$ is consistent if

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

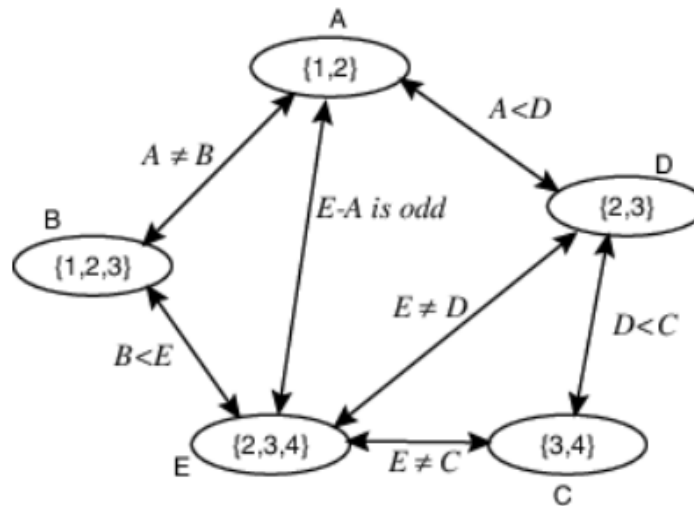


- Arc consistency detects failure earlier than forward checking.
- For some problems, it can speed up search enormously.
- For others, it may slow the search due to computational overheads.

Variable Elimination

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

Variable Elimination

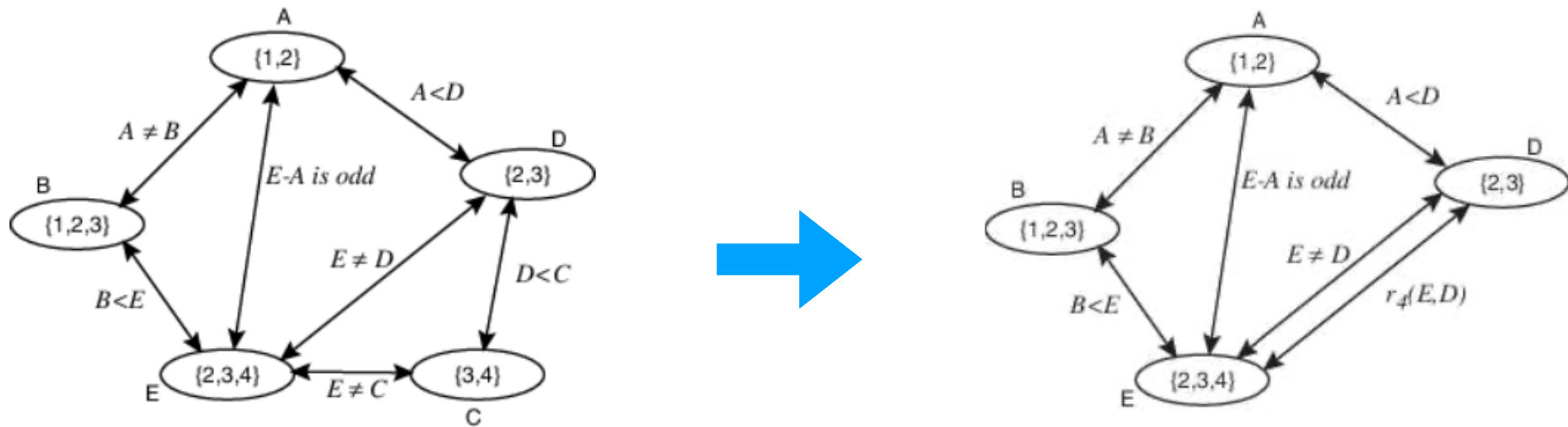


Variables: A, B, C, D, E

Domains: $A = \{1,2\}$, $B = \{1,2,3\}$, $C = \{3,4\}$, $D = \{2,3\}$, $E = \{2,3,4\}$

Constraints: $A \neq B$, $E \neq C$, $E \neq D$, $A < D$, $B < E$, $D < C$, $E - A \text{ is odd}$

Variable Elimination

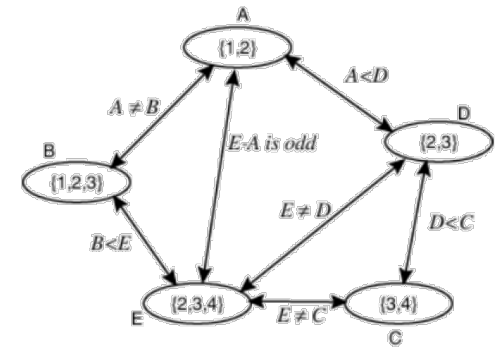


- Eliminates variable, one by one
- Replace them with constraints on adjacent variable

Variable Elimination Example

1. Select a variable X
2. Join the constraints in which X appears

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

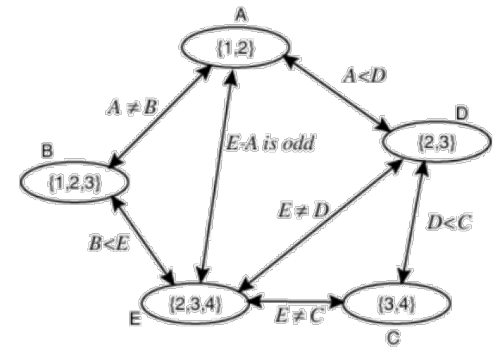


Variable Elimination Example

1. Select a variable X
2. Join the constraints in which X appears

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

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



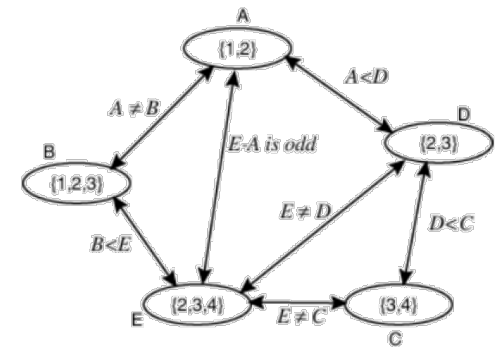
Variable Elimination Example

1. Select a variable X
2. Join the constraints in which X appears

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

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

$r_3 : r_1 \bowtie r_2$	C	D	E
	3	2	2
	3	2	4
	4	2	2
	4	2	3
	4	3	2
	4	3	3



Variable Elimination Example

1. Select a variable X
2. Join the constraints in which X appears
3. Project join onto its variables other than X (forming r_4)

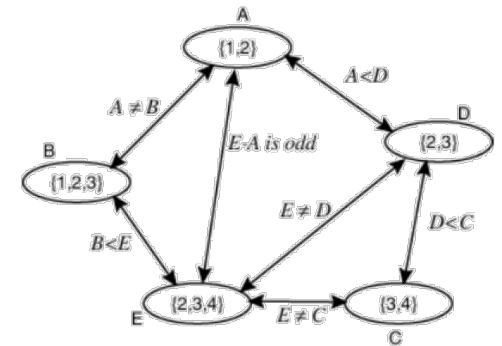
$r_1 : C \neq E$	C	E
	3	2
	3	4
	4	2
	4	3

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

$r_3 : r_1 \bowtie r_2$	C	D	E
	3	2	2
	3	2	4
	4	2	2
	4	2	3
	4	3	2
	4	3	3

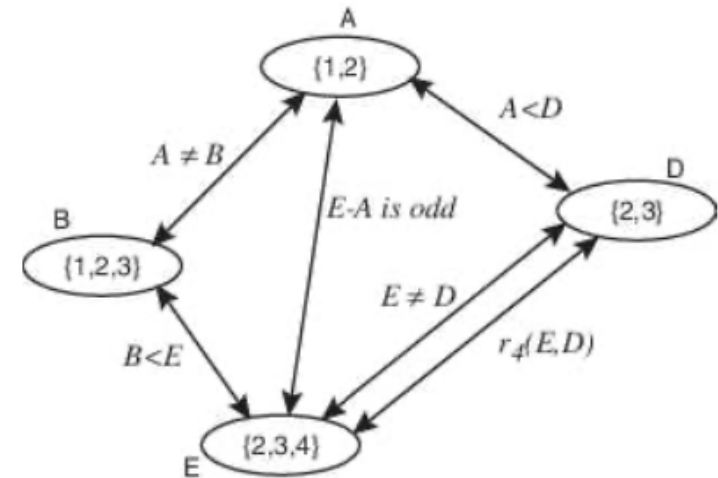
$r_4 : \pi_{\{D,E\}} r_3$	D	E
	2	2
	2	3
	2	4
	3	2
	3	3

↪ new constraint



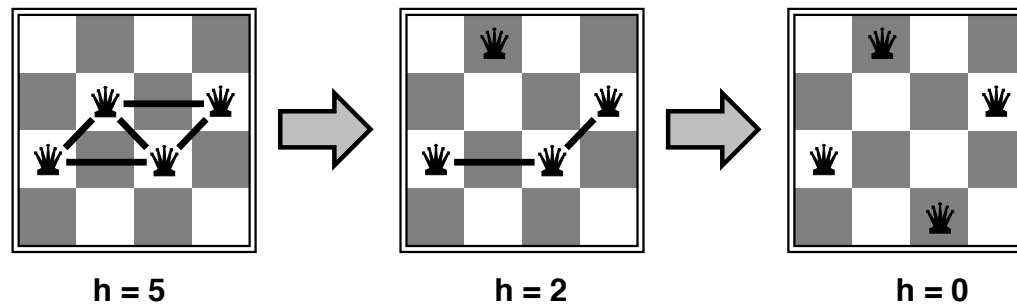
Variable Elimination Example

1. Select a variable X
2. Join the constraints in which X appears, forming constraint R_1
3. Project R_1 onto its variables other than X , forming R_2
4. Replace all of the constraints in which X appears by R_2
5. Recursively solve the simplified problem, forming R_3
6. Return R_1 joined with R_3



Local Search

There is another class of algorithms for solving CSP's, called “[Iterative Improvement](#)” or “Local Search”.

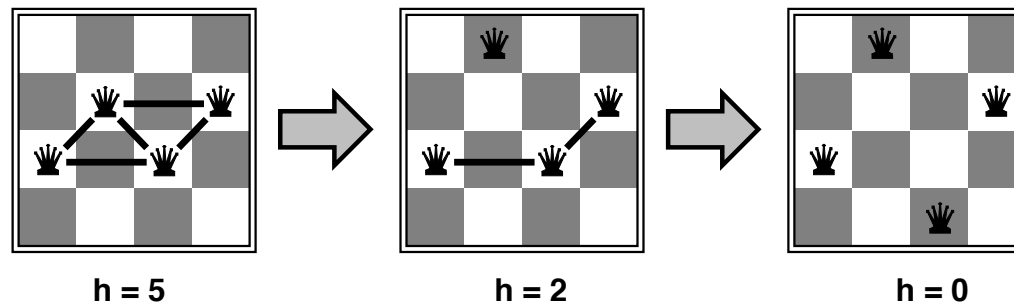


Local Search

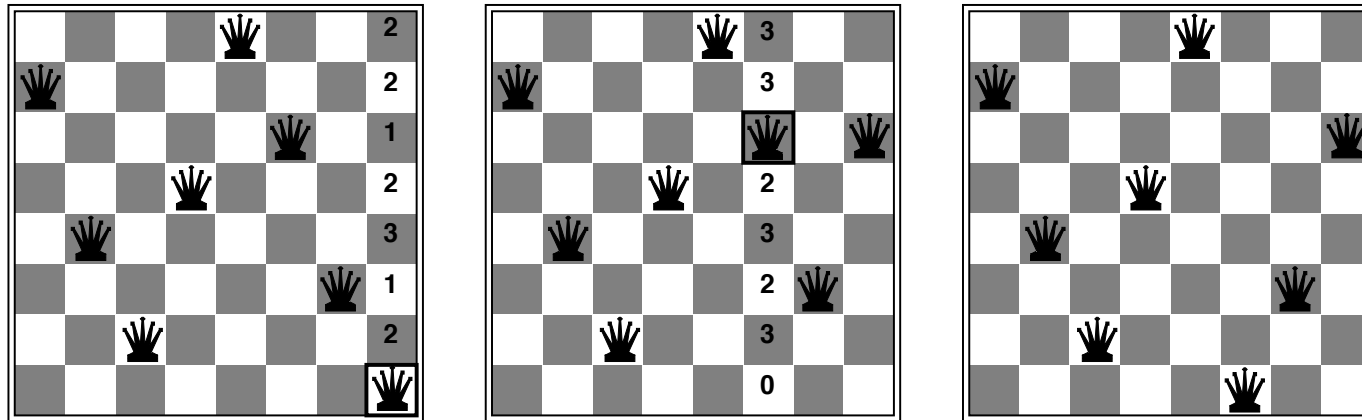
There is another class of algorithms for solving CSP's, called “[Iterative Improvement](#)” or “Local Search”.

- [Iterative Improvement](#)

- assign all variables randomly in the beginning (thus violating several 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

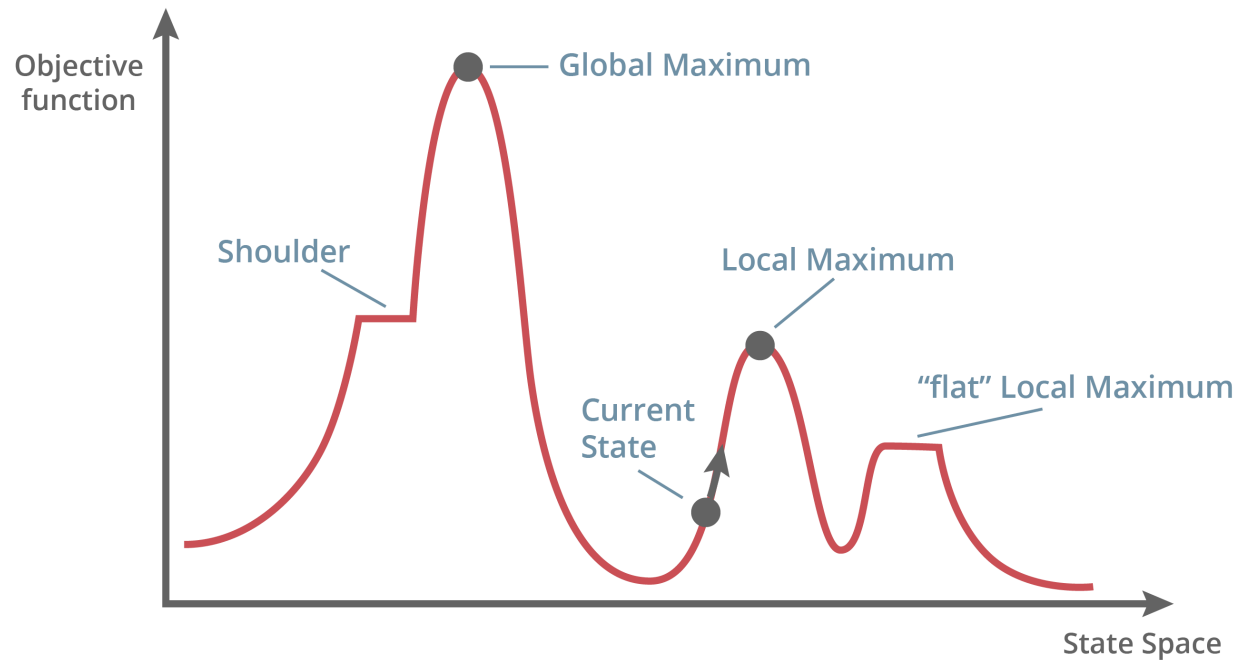


Hill-climbing by min-conflicts



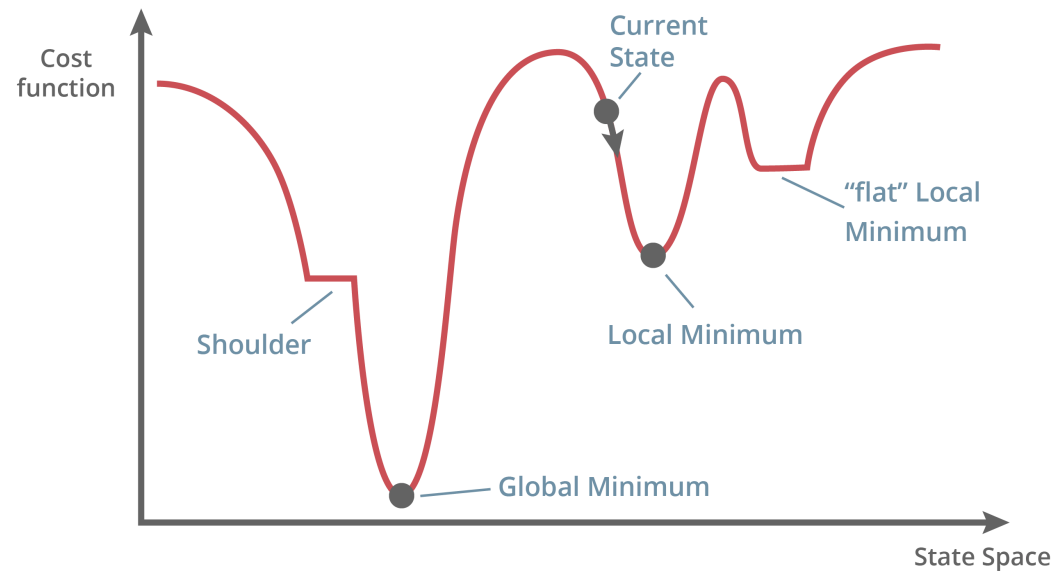
- Variable selection: randomly select any conflicted variable
- Value selection by min-conflicts heuristic
 - choose value that violates the fewest constraints

Flat regions and local optima



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

Inverted View



When we are minimising 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

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

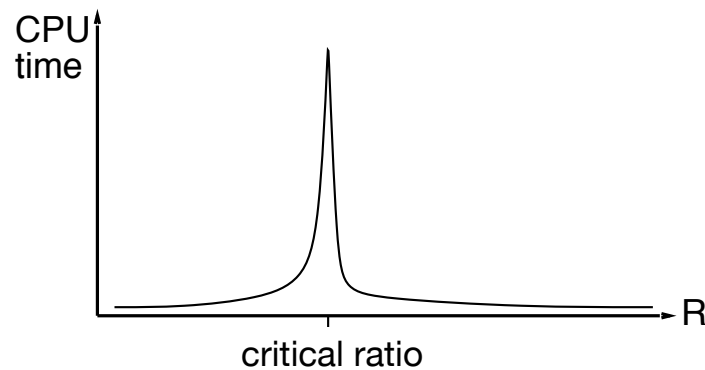
where T is a “temperature” parameter.

- Reduces to ordinary hill climbing when $T = 0$
- Becomes totally random search as $T \rightarrow \infty$
- Sometimes, we gradually decrease the value of T during the search

Phase Transition in CSP's

- 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).
- 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

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



Summary

- CSPs are a special kind of search 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
- Simulated Annealing can help to escape from local optima