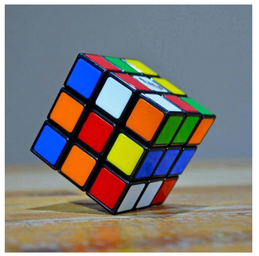
Constraint Satisfaction Problems

COMP3411/9814: Artificial Intelligence

2023

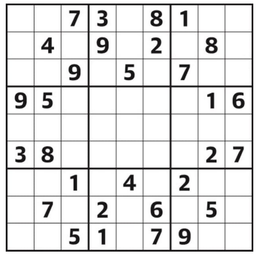
Path Search vs Constraint Satisfaction

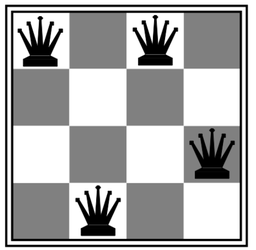


Difference between path search problems and CSPs

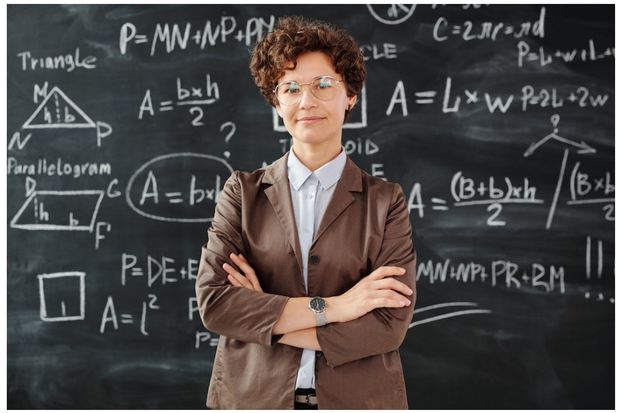
- Path Search Problems (e.g. Delivery Robot, Rubik's Cube)
  - Knowing the final state is easy
  - Difficult part is how to get there

- Constraint Satisfaction Problems (e.g. Sudoku, n-Queens)
  - Difficult part is knowing the final state
  - How to get there is easy



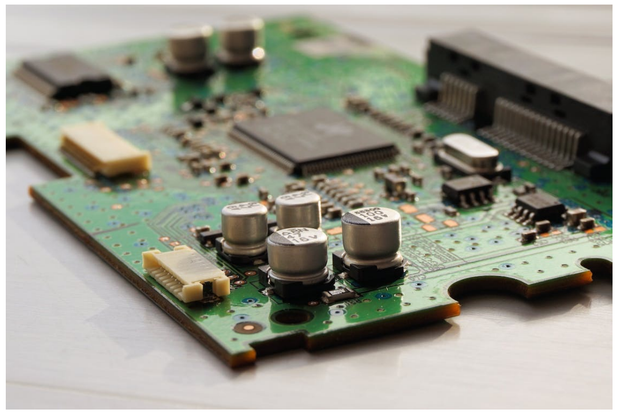


Constraint Satisfaction Problems



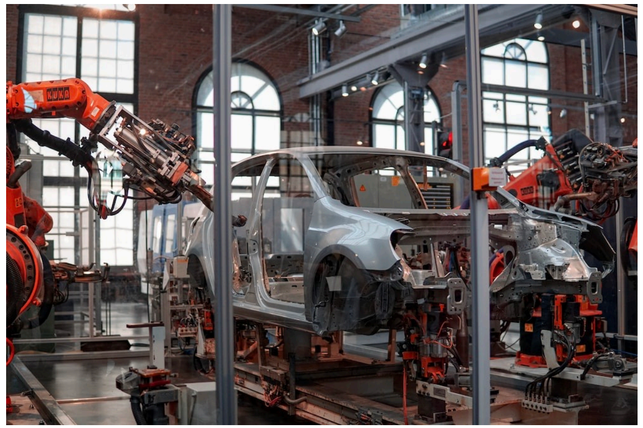
Assignment problems
(e.g. who teaches what class)

Timetabling problems
(e.g. which class is offered when and where?)



Hardware configuration
(e.g. minimise space for circuit layout)

Constraint Satisfaction Problems



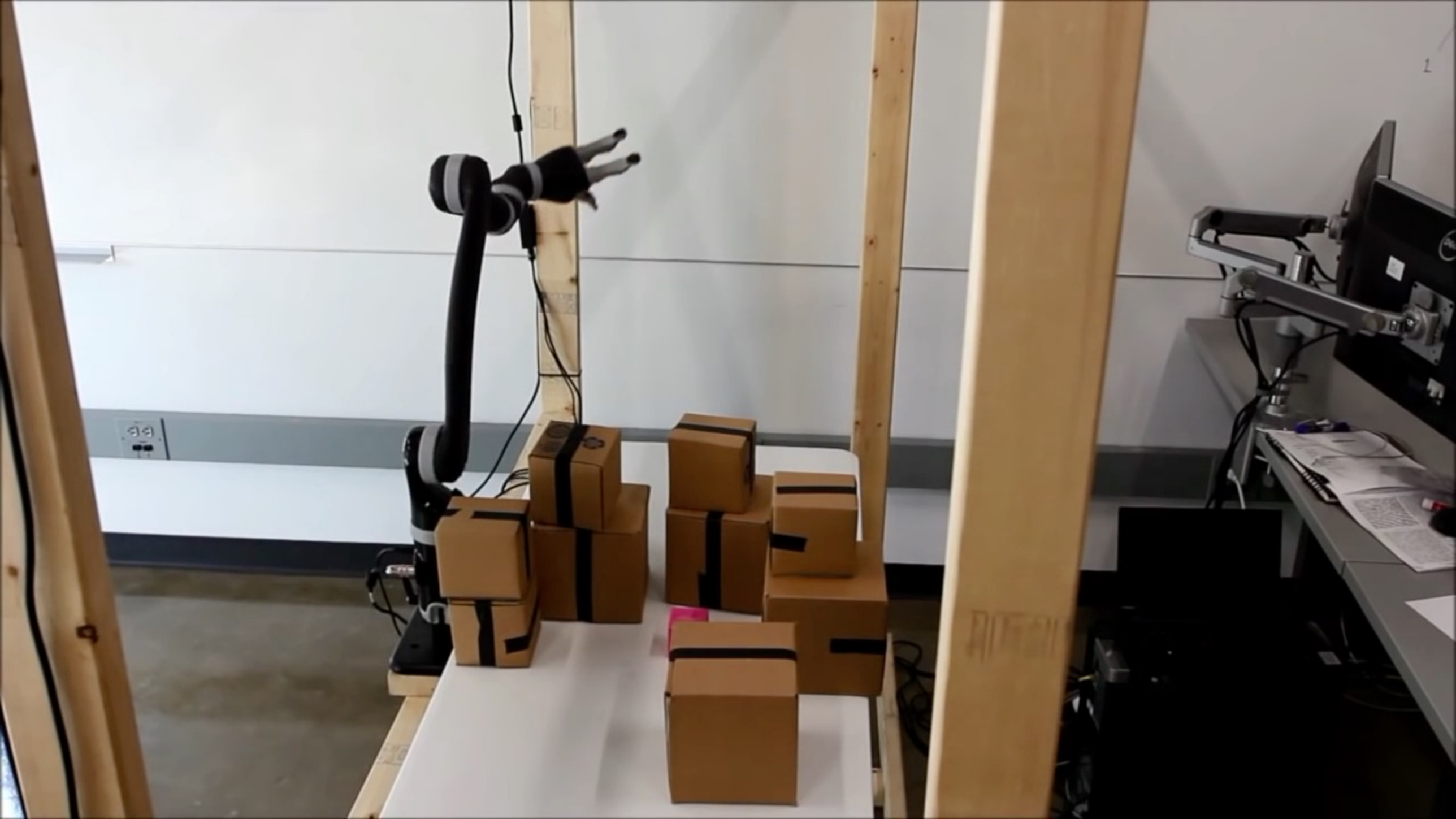
Factory scheduling
(optimise assignment
of jobs to machines)

Gate assignment
(assign gates to aircraft
to minimise transit)

Automated leather CNC cutting
(Cut leather to
minimise waste)

Constraint Satisfaction Problems

Closely related to optimisation problems



Lecture Overview

- Constraint Satisfaction Problems (CSPs) 
- CSP examples
- Varieties of CPSs
- Backtracking search and heuristic
- Forward checking and arc consistency
- Variable elimination
- Local search
- Domain splitting

Constraint Satisfaction Problems (CSPs)

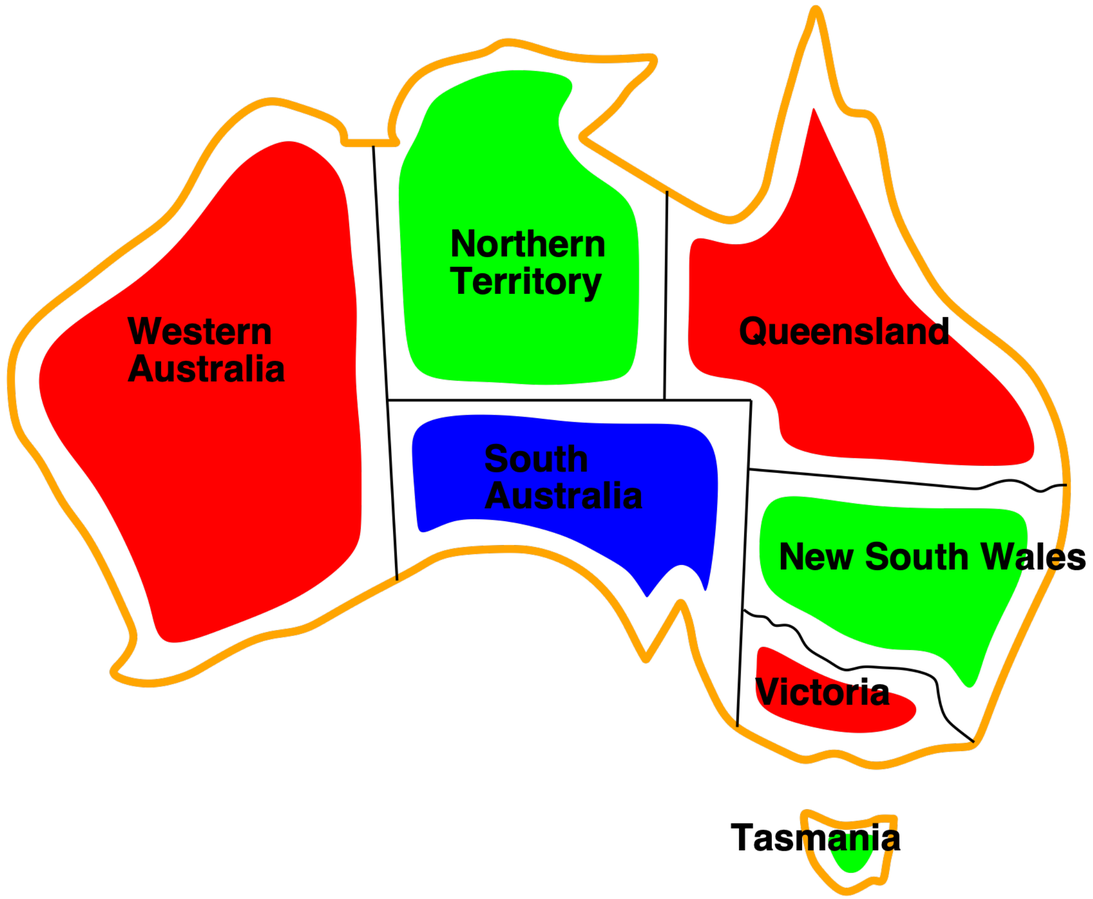
- Constraint Satisfaction Problems are defined by
  - a set of variables X_i,
  - each with a domain D_i of possible values,
  - and a set of constraints C that specify allowable combinations of values.

- 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.

Lecture Overview

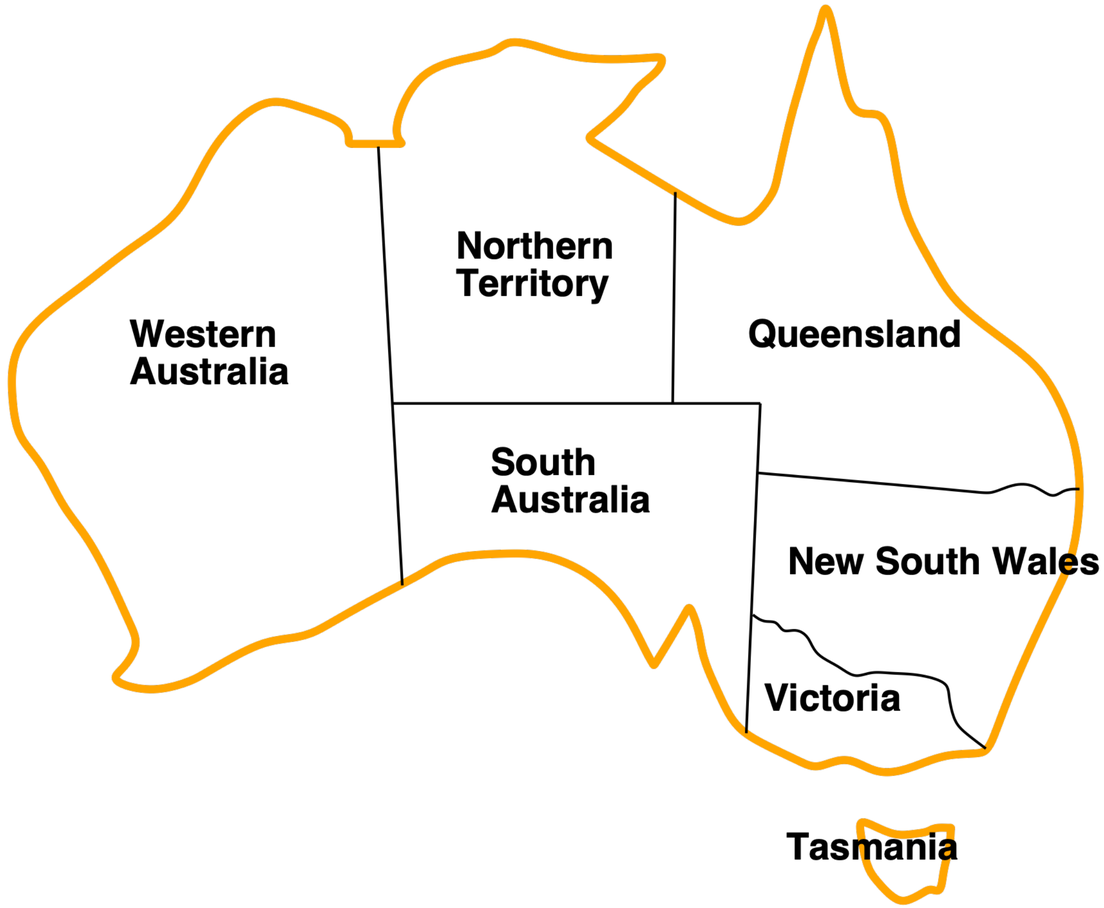
- Constraint Satisfaction Problems (CSPs)
- CSP examples
- Varieties of CPSs
- Backtracking search and heuristic
- Forward checking and arc consistency
- Variable elimination
- Local search
- Domain splitting

Example: Map-Colouring



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

Example: Map-Colouring



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

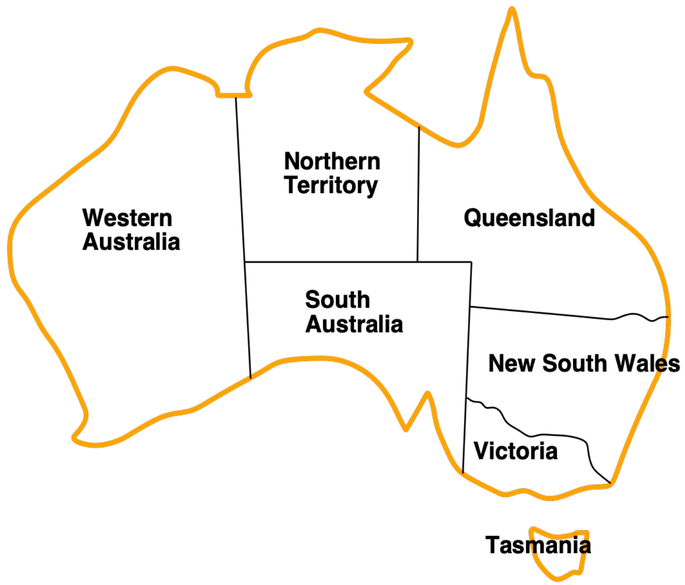
Domains: D_i = {red, green, blue}

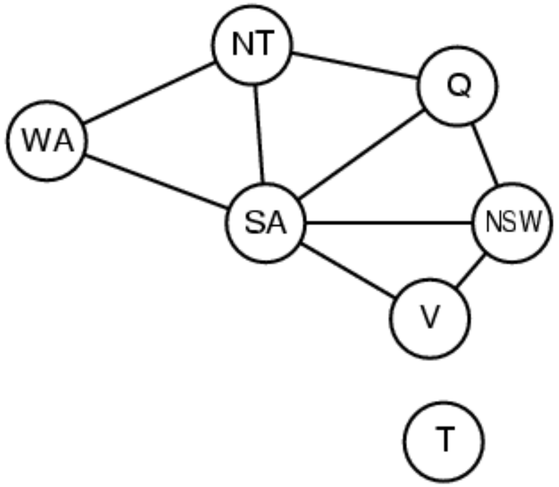
Constraints: adjacent regions must have different colours

e.g. WA≠ NT, etc.

Constraint graph

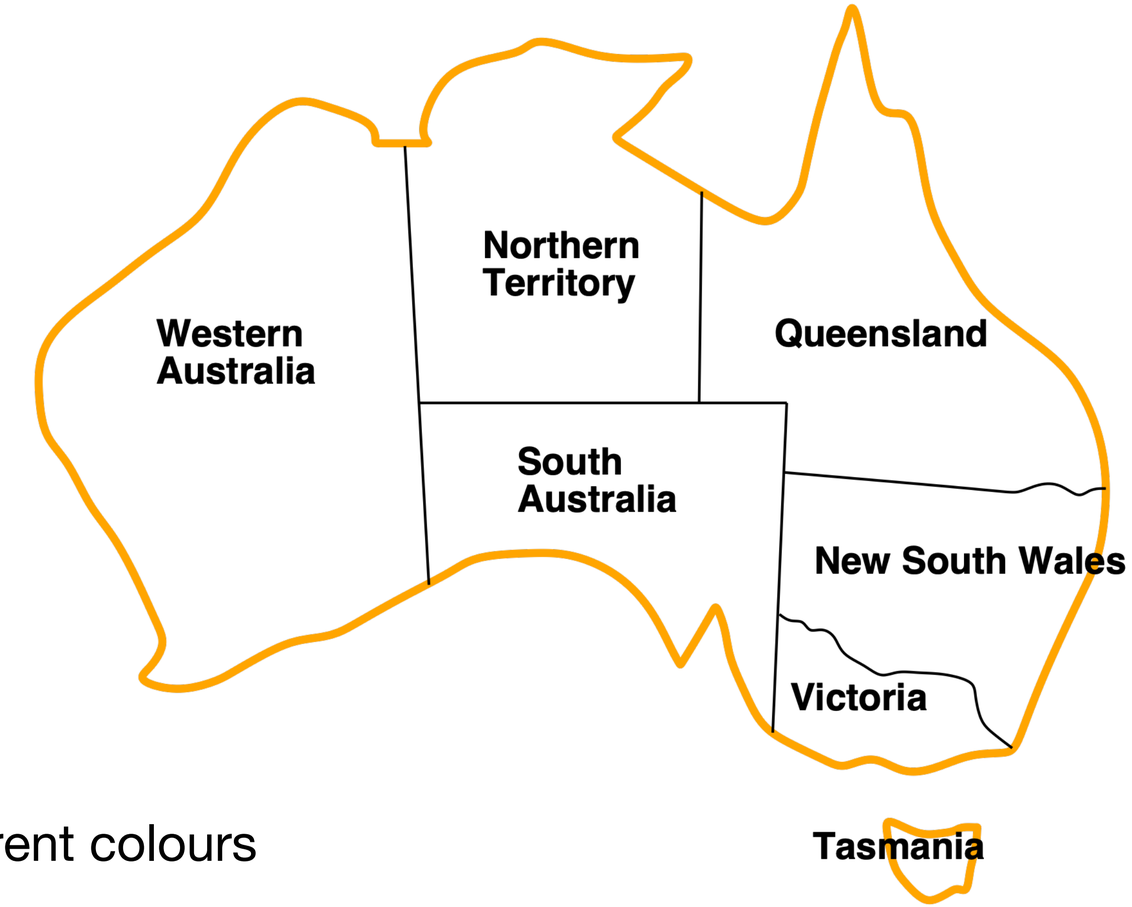
Constraint graph: nodes are variables, arcs are constraints





Binary CSP: each constraint relates two variables

Example: Map-Colouring



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

Domains: D_i = {red, green, blue}

Constraints: adjacent regions must have different colours

e.g. WA≠ NT, etc.

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

Example: Cryptarithmetic (exercise)

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

Variables:
?

Domains:
?

Constraints:
?

Example: Cryptarithmetic

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

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.

if there is a carry over, have to add to next column

Example: Cryptarithmetic

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

Variables: F T U W R O C_1 C_2 C_3

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

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

$O + O = R + 10 \cdot C_1$

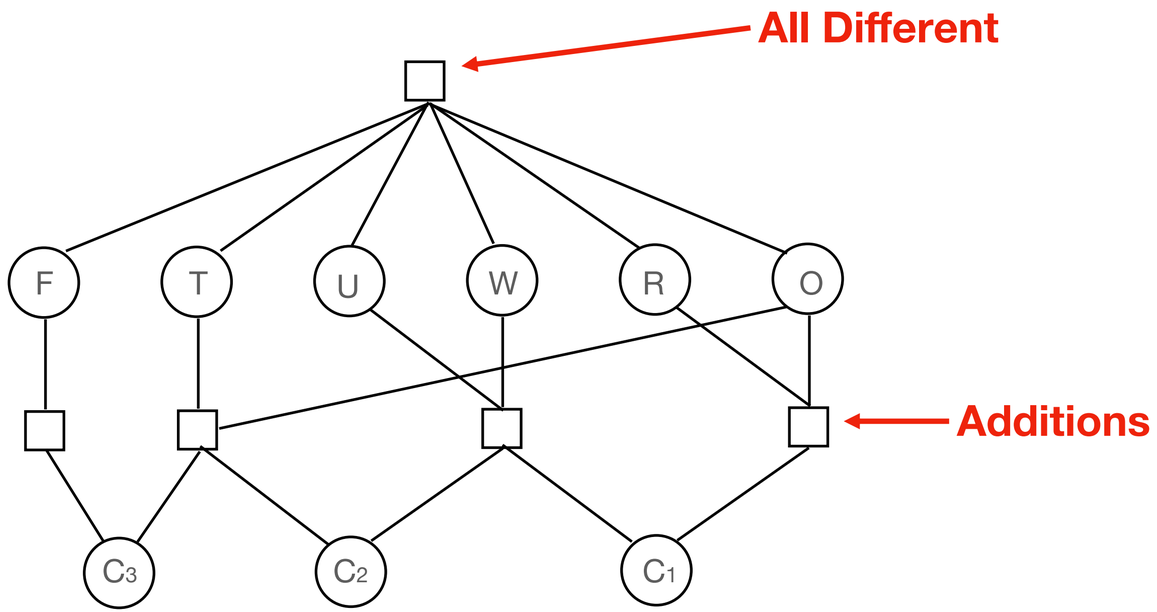
$C_1 + W + W = U + 10 \cdot C_2$

$C_2 + T + T = O + 10 \cdot C_3$

$C_3 = F$

Cryptarithmetic with Auxiliary Variables

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



Variables: F T U W R O C_1 C_2 C_3

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

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

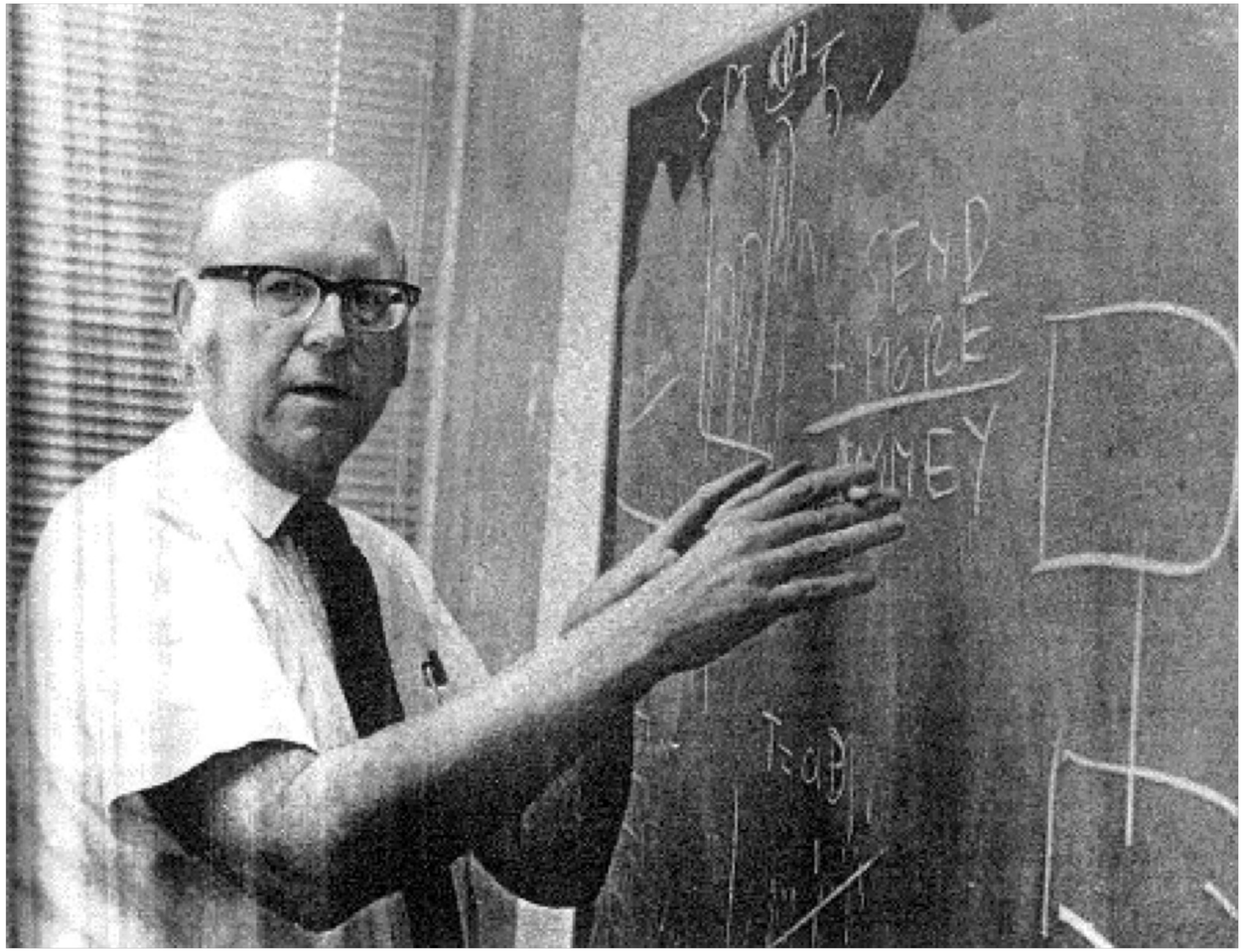
$O + O = R + 10 \cdot C_1$

$C_1 + W + W = U + 10 \cdot C_2$

$C_2 + T + T = O + 10 \cdot C_3$

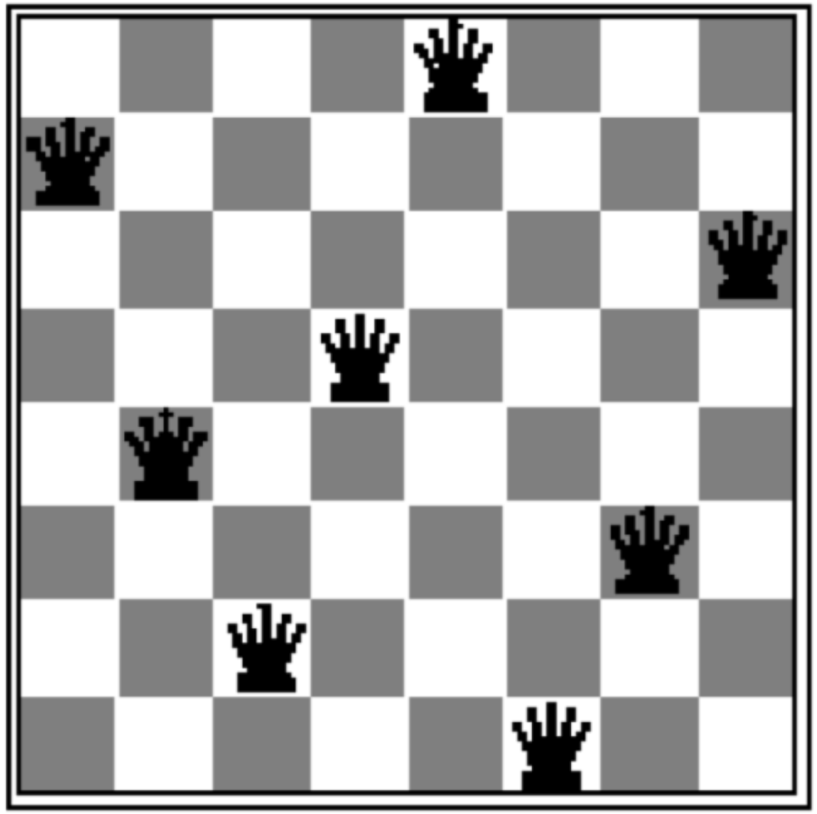
$C_3 = F$

Cryptarithmetic with Allen Newell



Book: Intended Rational Behavior

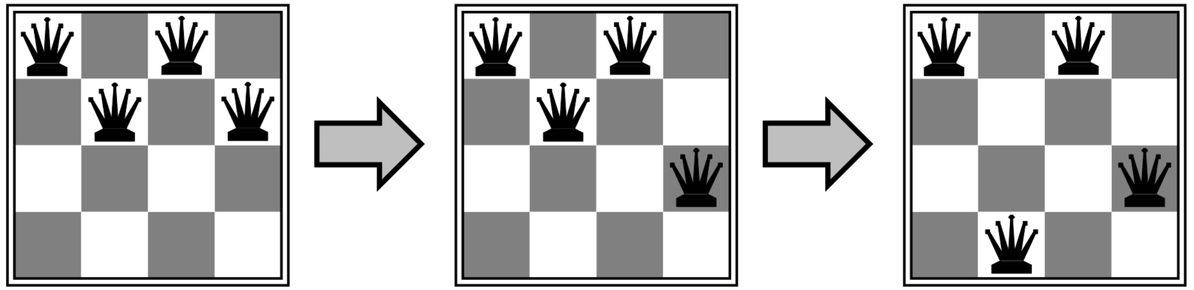
Example: n-Queens Puzzle



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. 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: $Q1$, $Q2$, $Q3$, $Q4$

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)

{1, 2, 1, 3}

Violates constraints because

$Q_1 = Q_3$ **and**

$|Q_1 - Q_2| = |i - j| = |1 - 2| = 1$

CSP Application - Factory scheduling

- An agent has 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 agent 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 bolding
  - 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: $\{1, 2, 3, 4\}$

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

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: $A,\ B, C$ - *v*ariables that represent the date of each activity

Domain of each variable is: $\{1, 2, 3, 4\}$

Constraint:

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

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

Lecture Overview

- Constraint Satisfaction Problems (CSPs)
- CSP examples 
- Varieties of CPSs
- Backtracking search and heuristic
- Forward checking and arc consistency
- Variable elimination
- Local search
- Domain splitting

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 *linear programming* 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

  $EndJob1 + 5 \leq StartJob3$

- Soft constraints (preferences)

  11am lecture is better than 8am lecture!

Lecture Overview

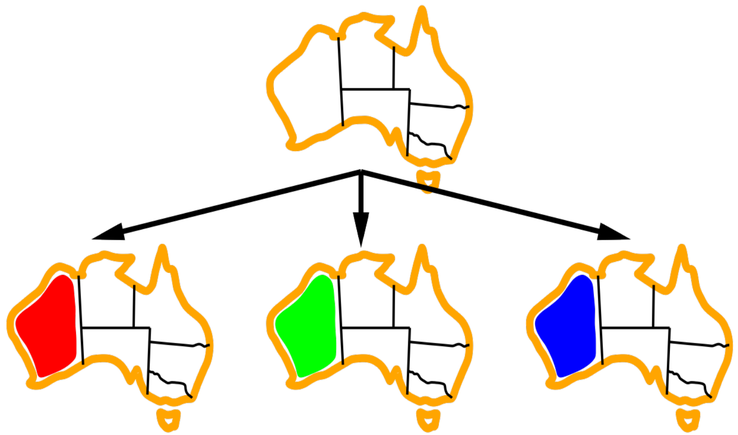
- Constraint Satisfaction Problems (CSPs)
- CSP examples
- Varieties of CPSs 
- Backtracking search and heuristic
- Forward checking and arc consistency
- Variable elimination
- Local search
- Domain splitting

Backtracking Search

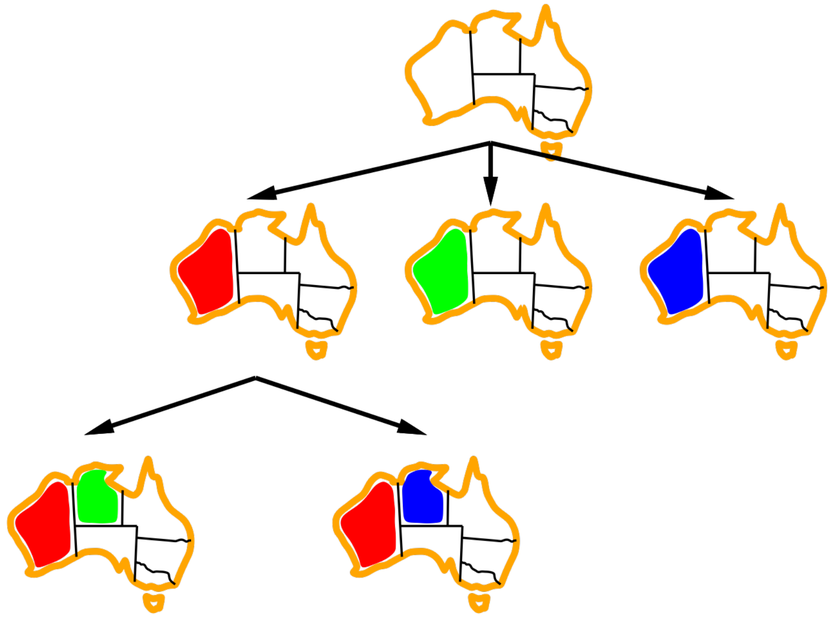
- CSPs can be solved by assigning values to variables one by one, in different combinations.
- Whenever a constraint is violated, go back to most recently assigned variable and assign it a new value.
- Can use Depth First Search, where states are defined by the values assigned so far:
  - Initial state: 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

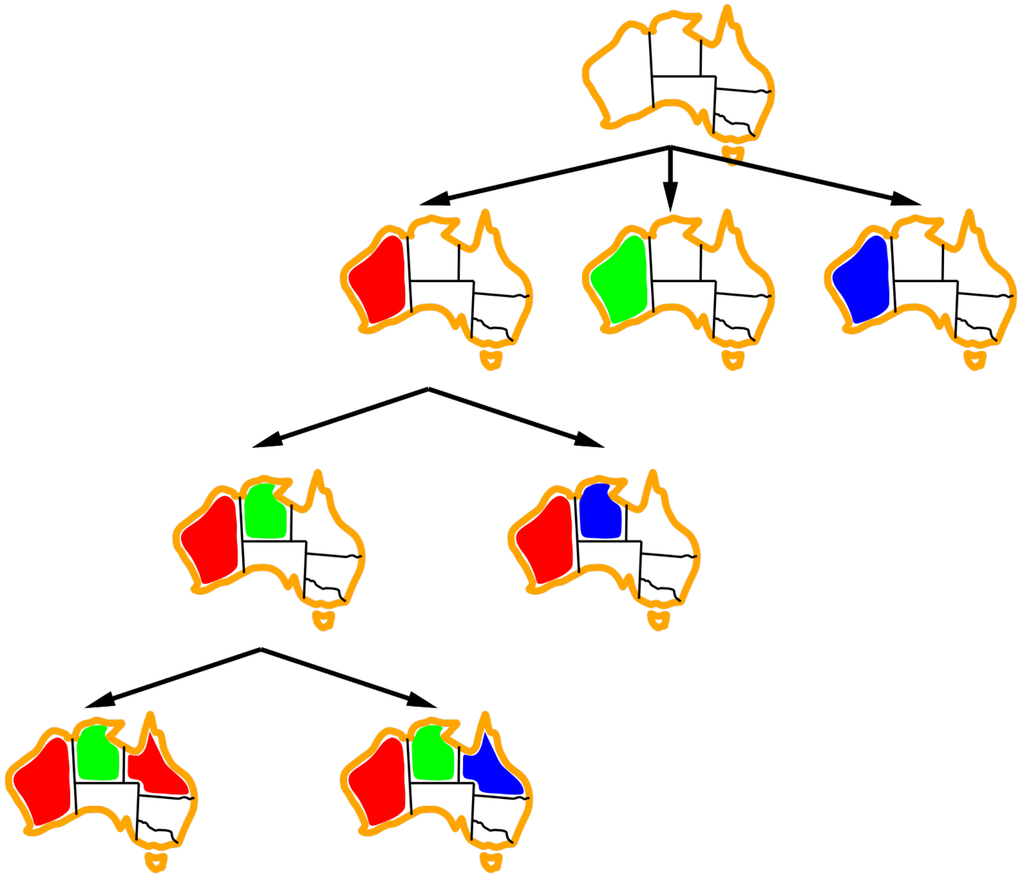
Backtracking example



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 ≈ 25

All Solutions by Backtracking

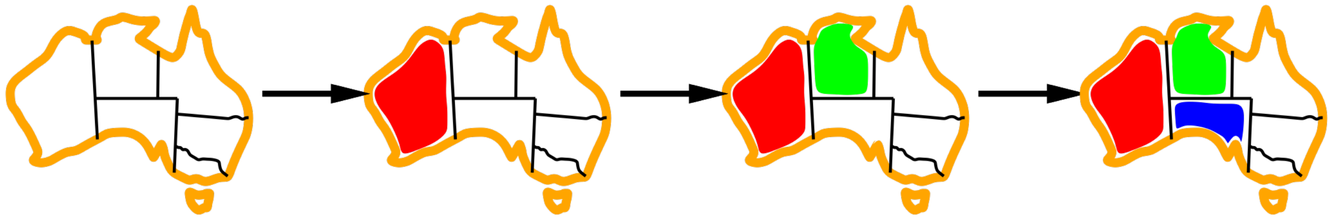
WA	NT	Q	NSW	V	SA	T
green	blue	green	blue	green	red	red
blue	green	blue	green	blue	red	red
red	blue	red	blue	red	green	red
blue	red	blue	red	blue	green	red
red	green	red	green	red	blue	red
green	red	green	red	green	blue	red
green	blue	green	blue	green	red	green
blue	green	blue	green	blue	red	green
red	blue	red	blue	red	green	green
blue	red	blue	red	blue	green	green
red	green	red	green	red	blue	green
green	red	green	red	green	blue	green
green	blue	green	blue	green	red	blue
blue	green	blue	green	blue	red	blue
red	blue	red	blue	red	green	blue
blue	red	blue	red	blue	green	blue
red	green	red	green	red	blue	blue
green	red	green	red	green	blue	blue

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

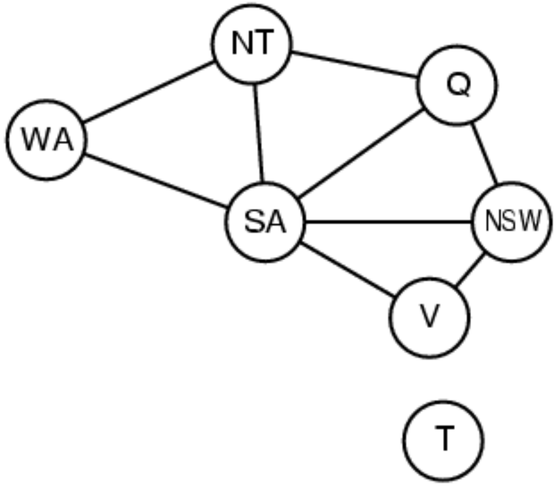
- Minimum Remaining Values (MRV)
  - choose the variable with the fewest legal remaining values
  - Most constrained variable

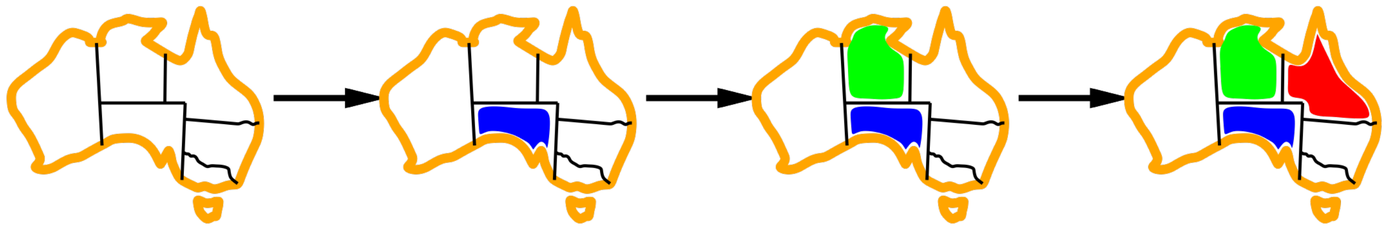


Degree Heuristic

- Tie-breaker among MRV variables

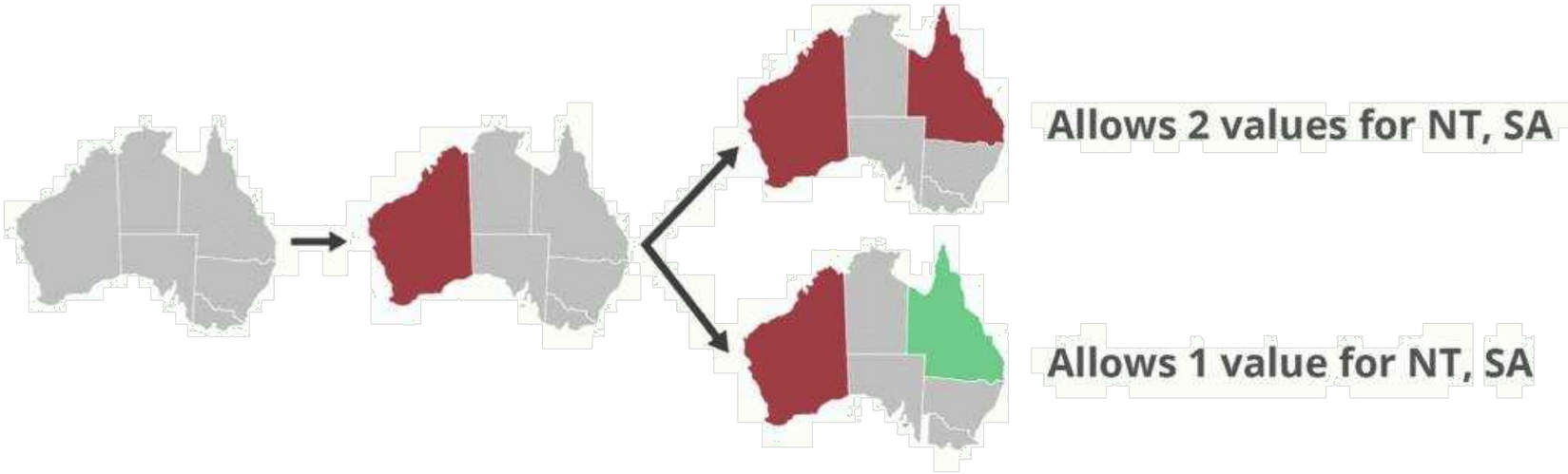
- Degree heuristic:
  - choose the variable with the most constraints on variables (i.e. most edges in graph)
  - If same degree, choose any one

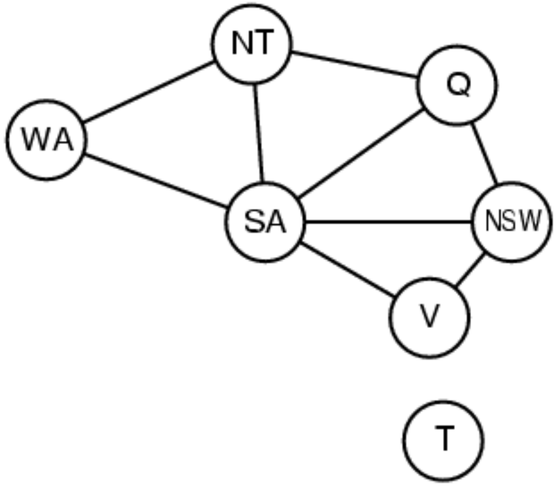




Least Constraining Value

- 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.

Lecture Overview

- Constraint Satisfaction Problems (CSPs)
- CSP examples
- Varieties of CPSs
- Backtracking search and heuristic 
- Forward checking and arc consistency
- Variable elimination
- Local search
- Domain splitting

Forward Checking

- Keep track of remaining legal values for unassigned variables

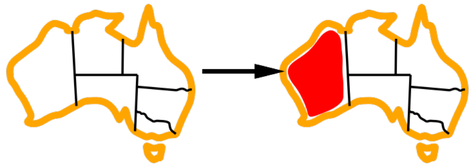
- Terminate search when any variable has no legal values
  - prune off that part of the search tree, and backtrack

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
  - prune off that part of the search tree, and backtrack



Forward Checking

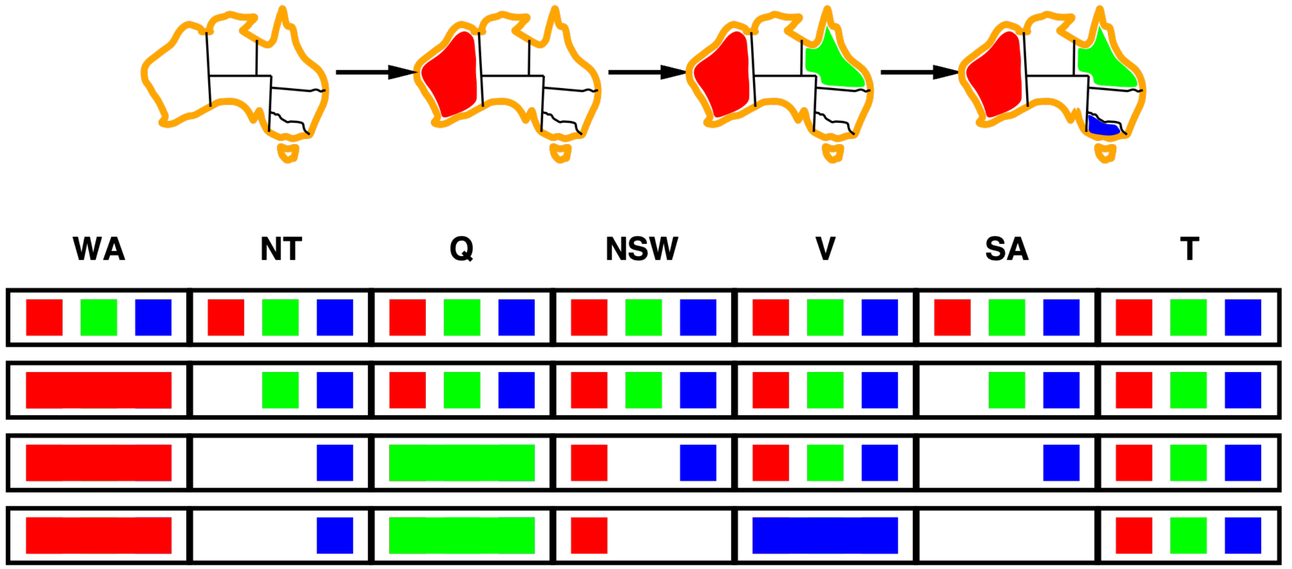
- Keep track of remaining legal values for unassigned variables

- Terminate search when any variable has no legal values
  - prune off that part of the search tree, and backtrack

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



Fail

Constraint Propagation

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

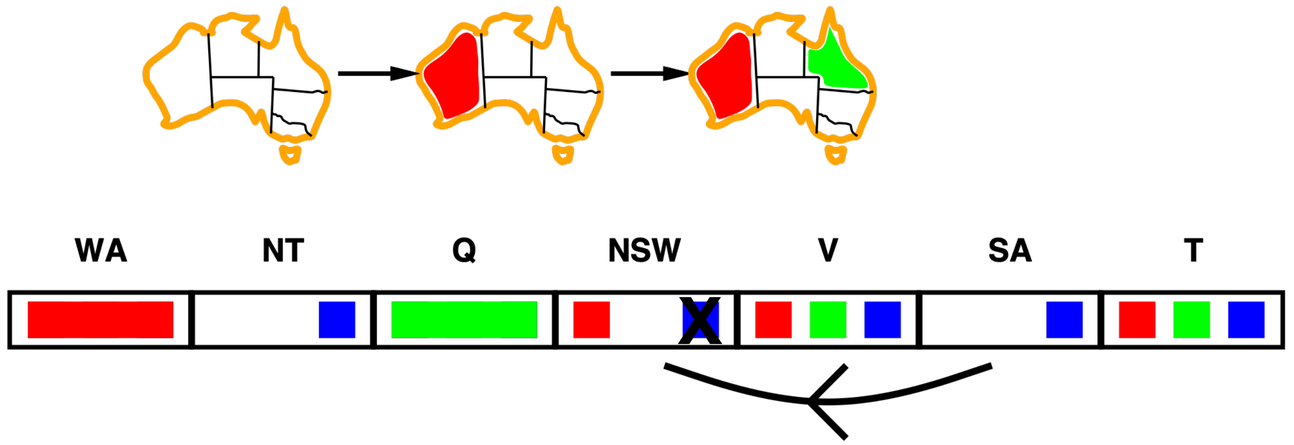
NT and SA cannot both be blue

Constraint propagation repeatedly enforces constraints locally

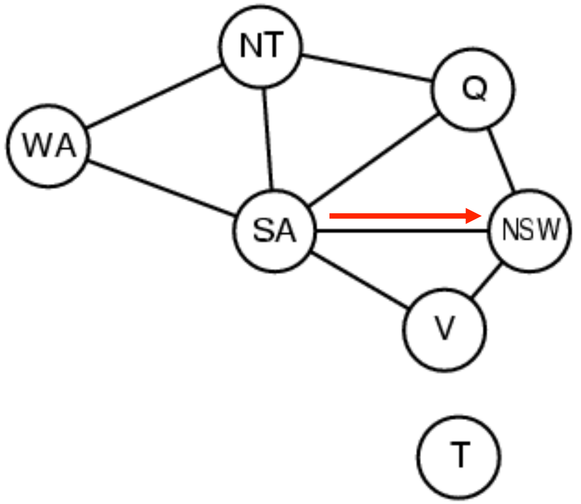
Arc Consistency

$X \rightarrow Y$ is consistent if

for every value x of X there is some allowed y



Only possible value fo SA is blue, so NSW can't blue

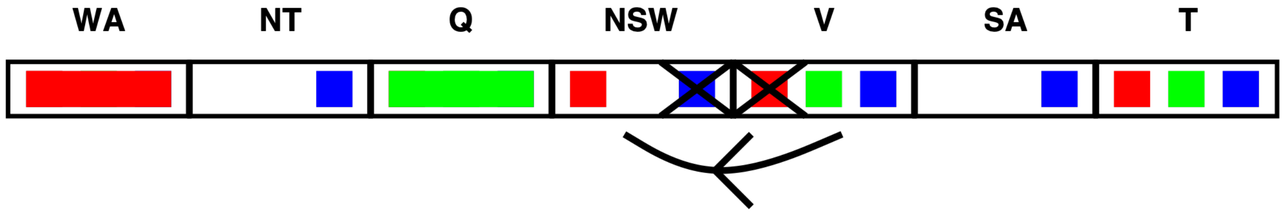


Propagate arc consistency on the graph

Arc Consistency

$X \rightarrow Y$ is consistent if

for every value x of X there is some allowed y

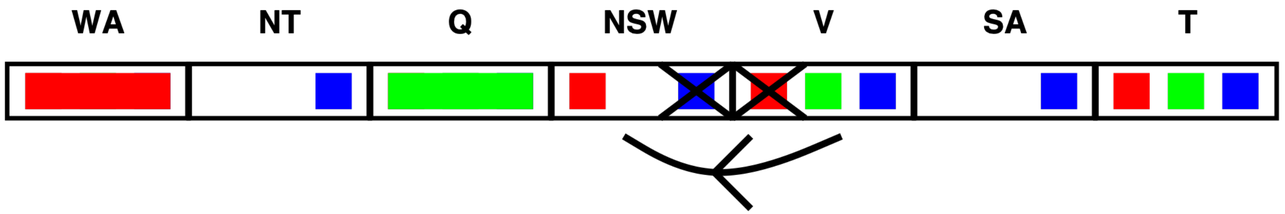


- If X loses a value, neighbours of X need to be rechecked.
- Since *NSW* can only be red now, *V* cannot be red

Arc Consistency

$X \rightarrow Y$ is consistent if

for every value x of X there is some allowed y

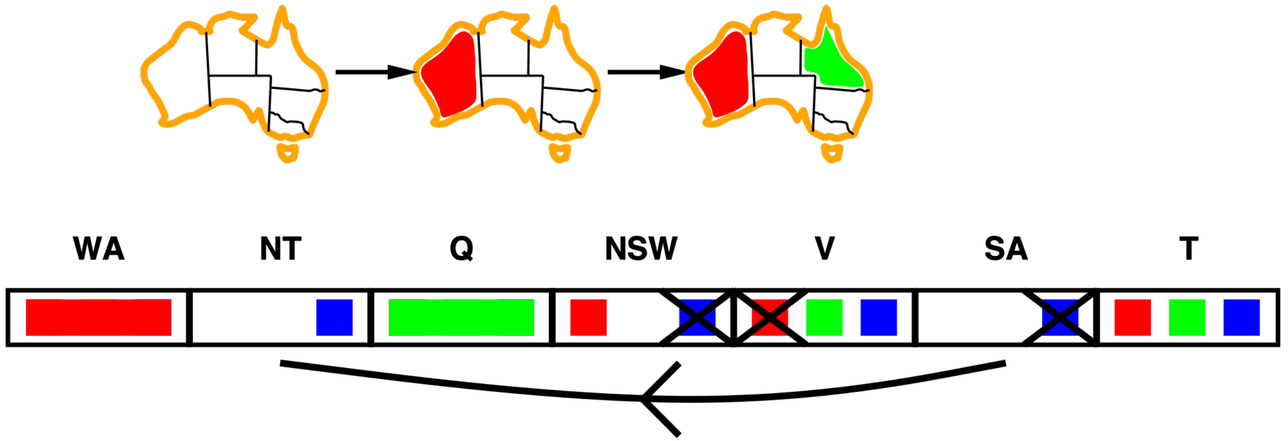


- If X loses a value, neighbours of X need to be rechecked.
- Arc consistency detects failure earlier than forward checking
- Can be run as a preprocessor after each assignment

Arc Consistency

$X \rightarrow Y$ is consistent if

for every value x of X there is some allowed y

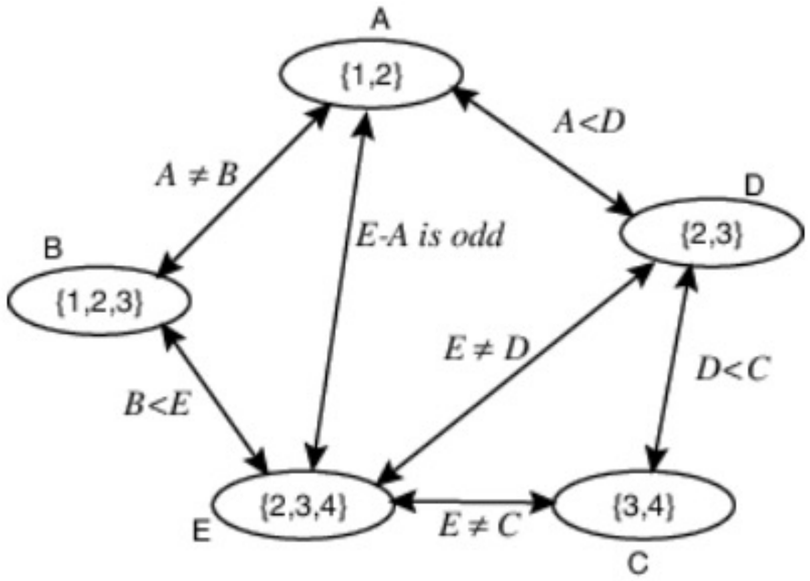


- Arc consistency detects failure earlier than forward checking.
- For some problems, it can speed up search enormously.
- For others, it may slow the search due to computational overheads.

Lecture Overview

- Constraint Satisfaction Problems (CSPs)
- CSP examples
- Varieties of CPSs
- Backtracking search and heuristic
- Forward checking and arc consistency 
- Variable elimination
- Local search
- Domain splitting

Variable Elimination

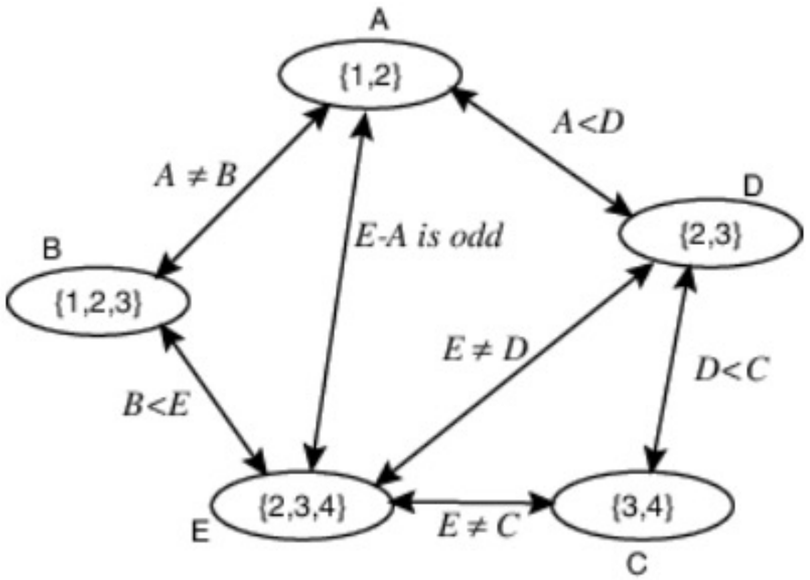


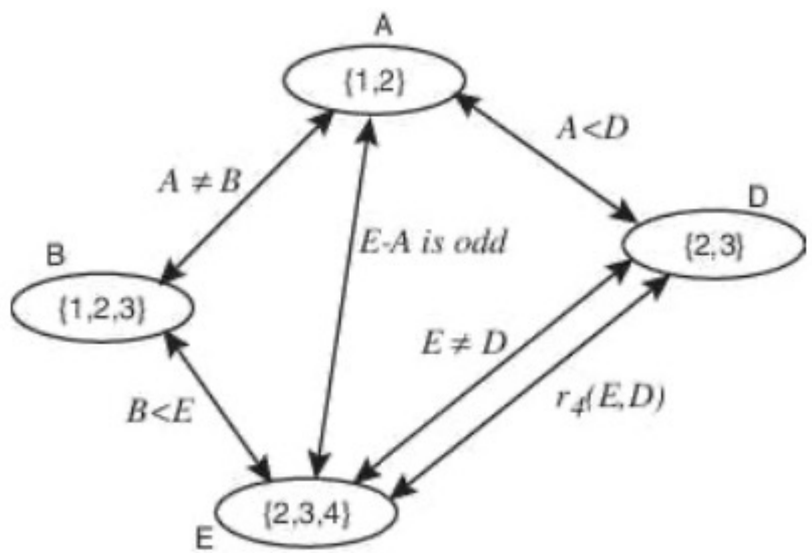
Variables: A, B, C, D, E

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

Constraints: A ≠ B, E ≠ C, E ≠ D, A < D, B < E, D < C, E-A is odd

Variable Elimination



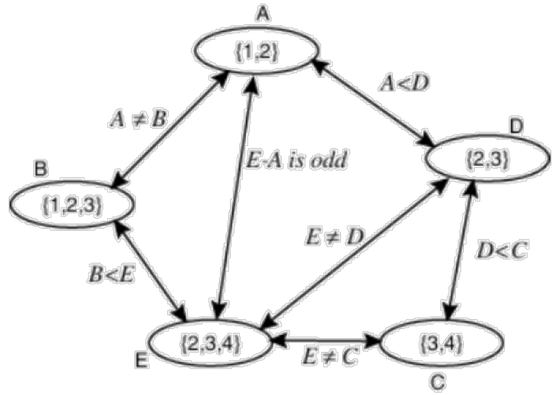


- Eliminates variable, one by one
- Replace them with constraints on adjacent variables

Variable Elimination Example

1. Select a variable X
2. Enumerate constraints

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

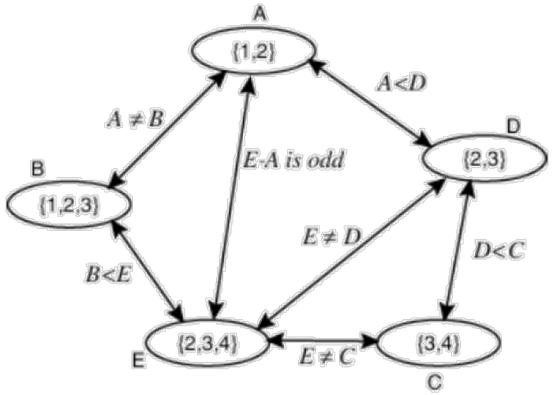


Variable Elimination Example

1. Select a variable X
2. Enumerate constraints

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

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



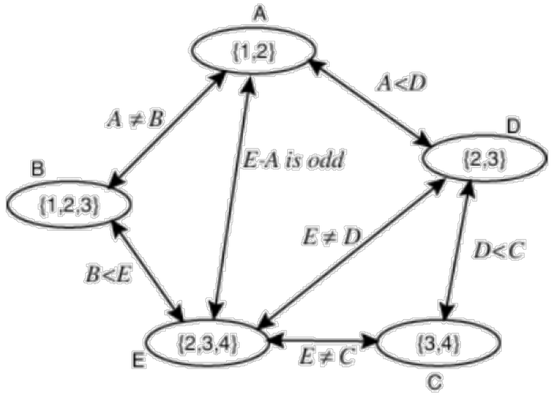
Variable Elimination Example

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

$r_3 : r_1 \bowtie r2$	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)

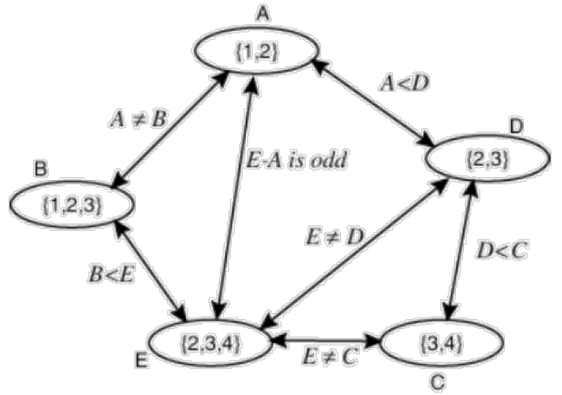
$r_1 : C \neq E$	C	E
	3	2
	3	4
	4	2
	4	3

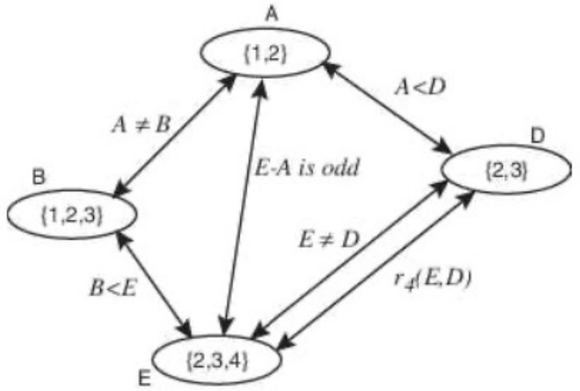
$r_2 : C > D$	C	D
	3	2
	4	2
	4	3

$r_3 : r_1 \bowtie r2$	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

↪ new constraint

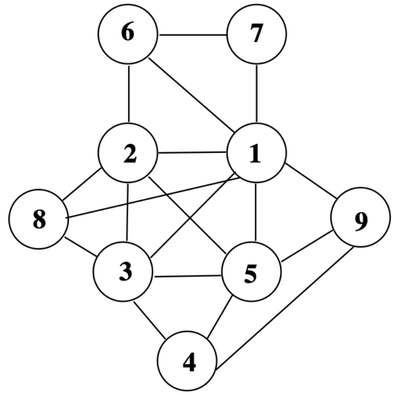




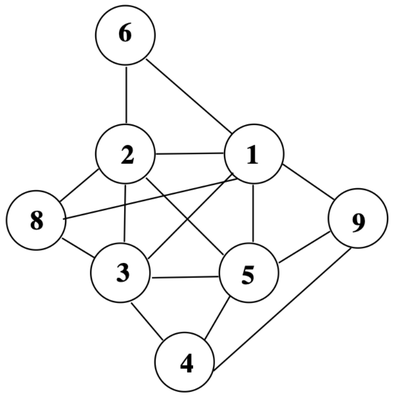
To generate one or all solutions, the algorithm remembers the joined relation *C*, *D*, *E* to construct a solution that involves *C* from a solution to the reduced network.

Variable Elimination

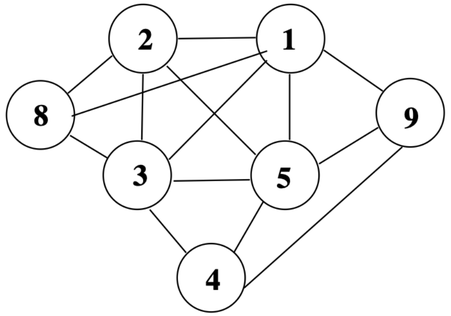
1. Select a variable X
2. Join the constraints in which X appears, forming constraint R1
3. Project R1 onto its variables other than X, forming R2
4. Replace all of the constraints in which X appears by R2
5. Recursively solve the simplified problem, forming R3
6. Return R1 joined with R3



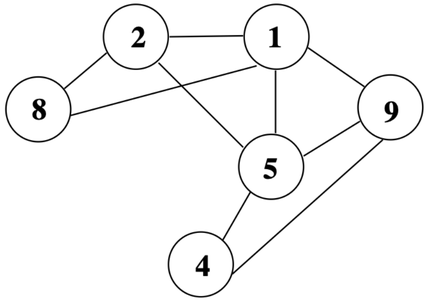
a) Initial constraint graph



b) After eliminating X7



c) After eliminating X6

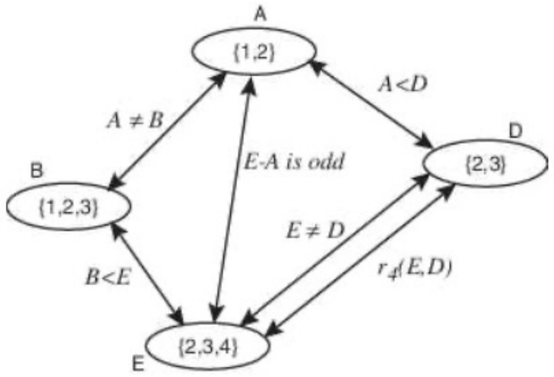


d) After branching in X3

Variable Elimination

1. If there is only one variable,

return the join of all the relations in the constraints

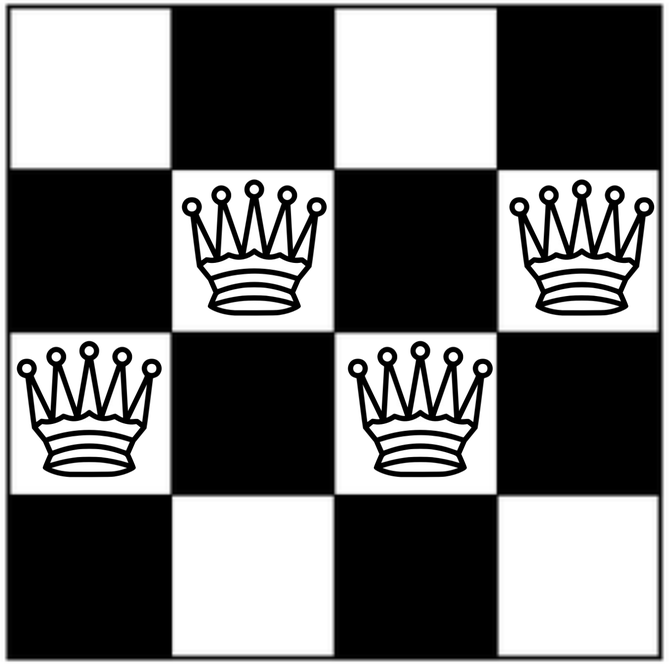


2. Otherwise
    1. Select a variable X
    2. Join the constraints in which X appears, forming constraint R1
    3. Project R1 onto its variables other than X, forming R2
    4. Replace all of the constraints in which X appears by R2
    5. Recursively solve the simplified problem, forming R3
    6. Return R1 joined with R3

Lecture Overview

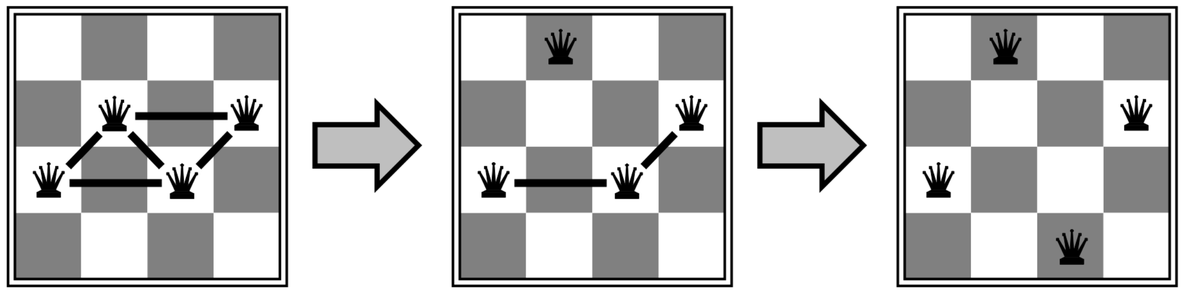
- Constraint Satisfaction Problems (CSPs)
- CSP examples
- Varieties of CPSs
- Backtracking search and heuristic
- Forward checking and arc consistency
- Variable elimination 
- Local search
- Domain splitting

Local search



Local Search

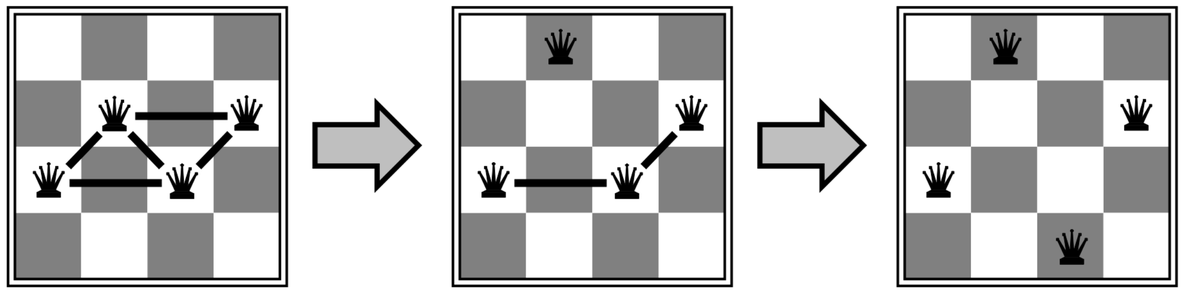
There is another class of algorithms for solving CSP's, called "Iterative Improvement" or "Local Search".



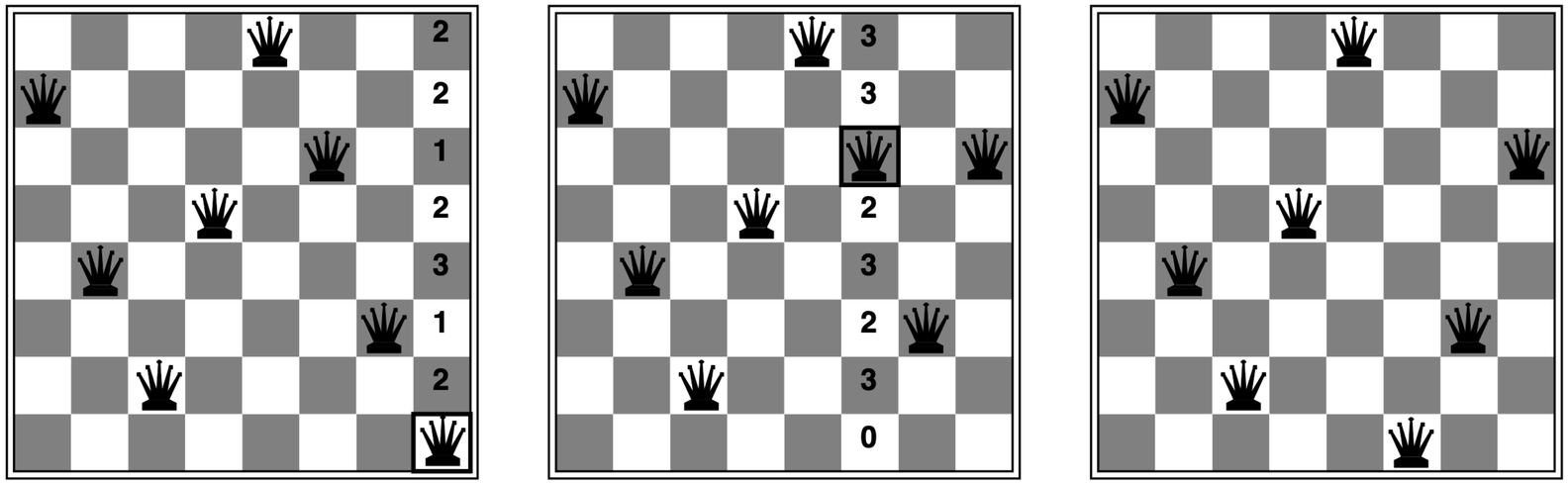
Local Search

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

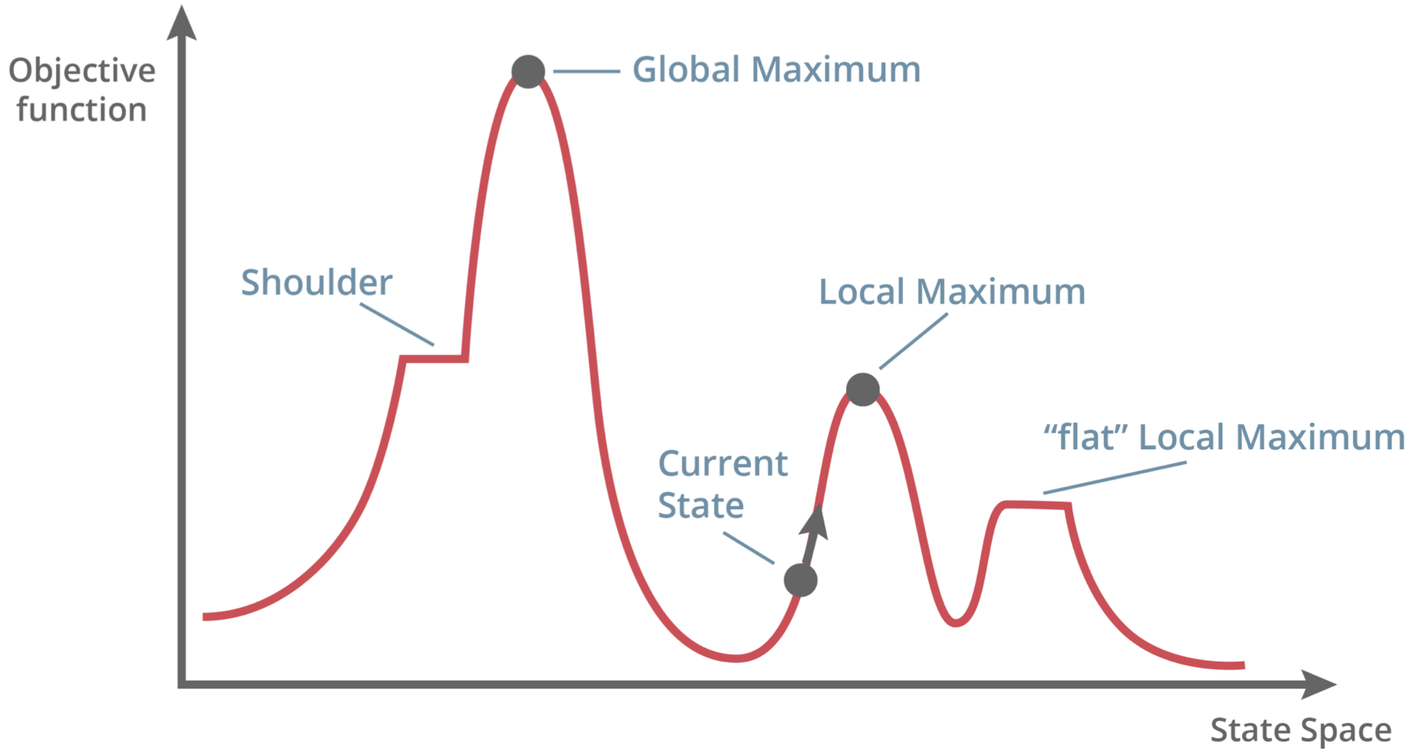


Hill-climbing by min-conflicts



- 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

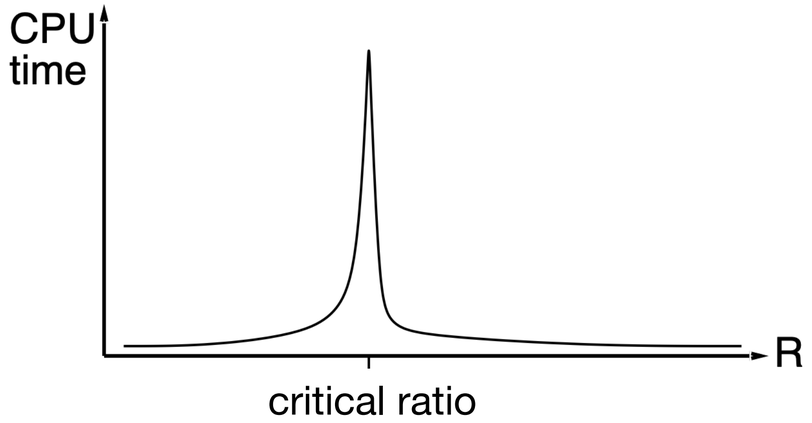


- May have to go sideways or backwards to make progress towards the solution

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

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



Lecture Overview

- Constraint Satisfaction Problems (CSPs)
- CSP examples
- Varieties of CPSs
- Backtracking search and heuristic
- Forward checking and arc consistency
- Variable elimination 
- Local search
- Domain splitting

Domain Splitting

Finding solution with AC and domain splitting

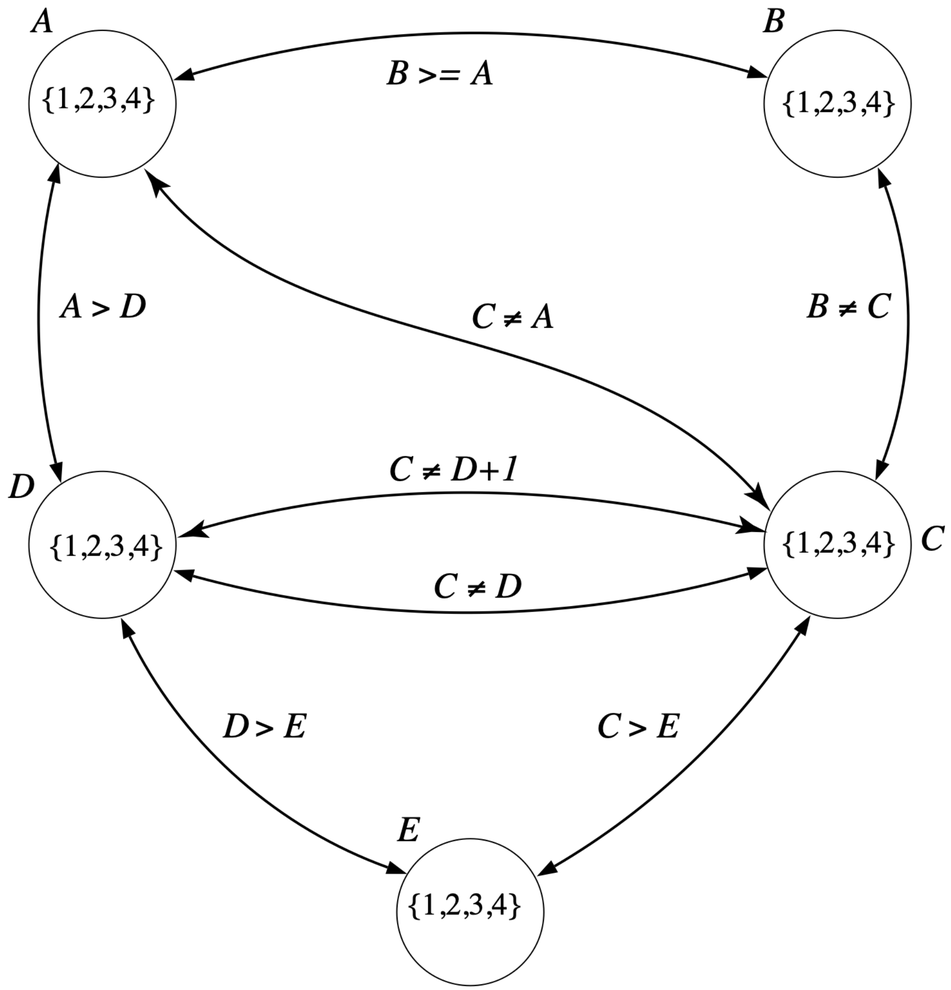
To solve a CSP:

- Simplify with arc-consistency
- If a domain is empty, return no solution
- If all domains have size 1,
  return solution found
  Else
  split a domain, and recursively solve each half.

Finding solution with AC and domain splitting

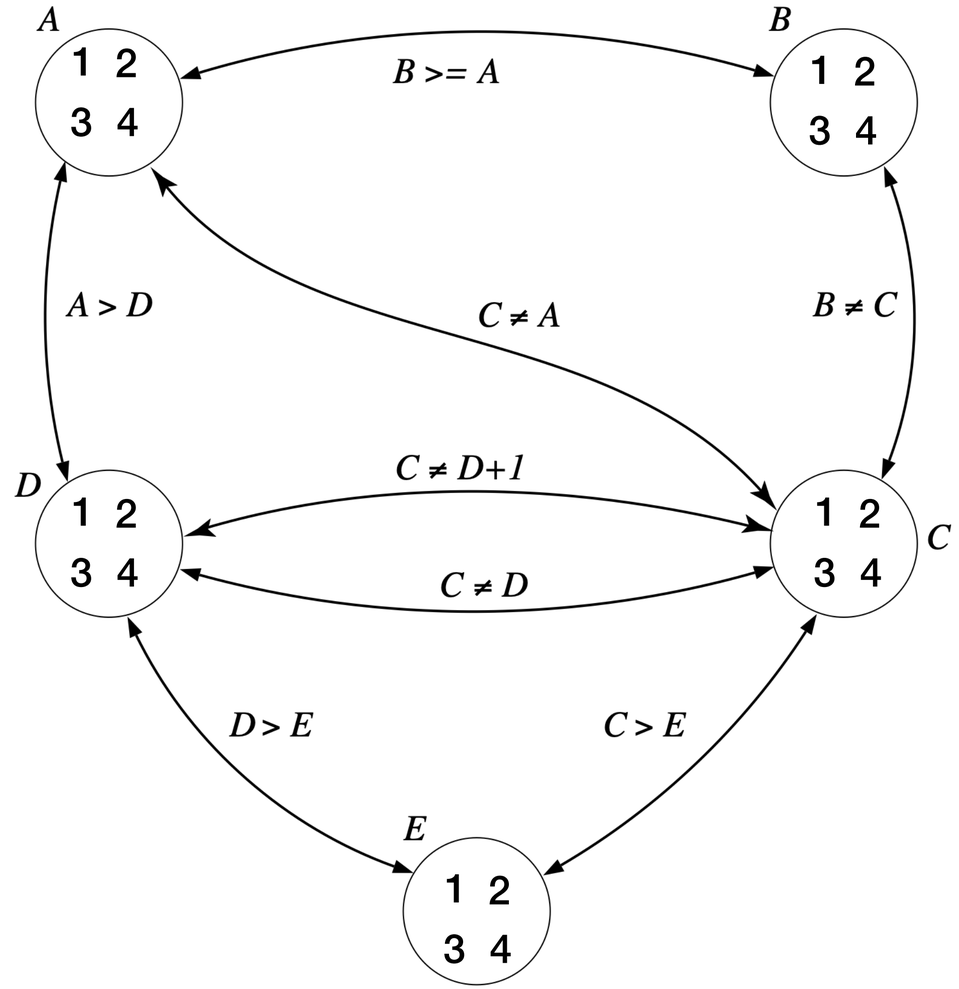
Solve (*CSP, domains*) :
    simplify *CSP* with arc-consistency
    if one domain is empty:
        return False
    else if all domains have one element:
        return solution
    else:
        select variable X with domain D and $|D| > 1$
        partition D into D_1 and D_2
        re *Solve* (*CSP, domains* with *dom*(X) = D_1) **or**
            Solve (*CSP, domains* with *dom*(X) = D_2)

AC exercise



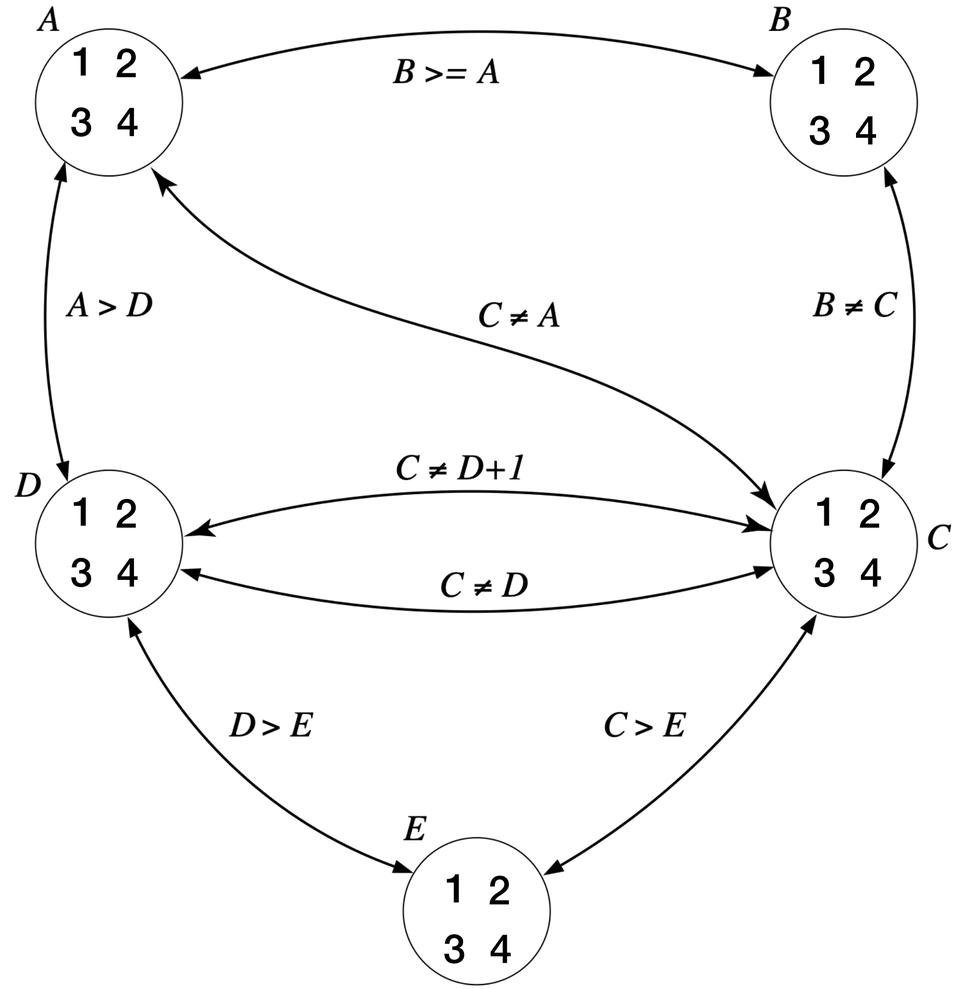
AC exercise

Arc	Relation	Values Removed



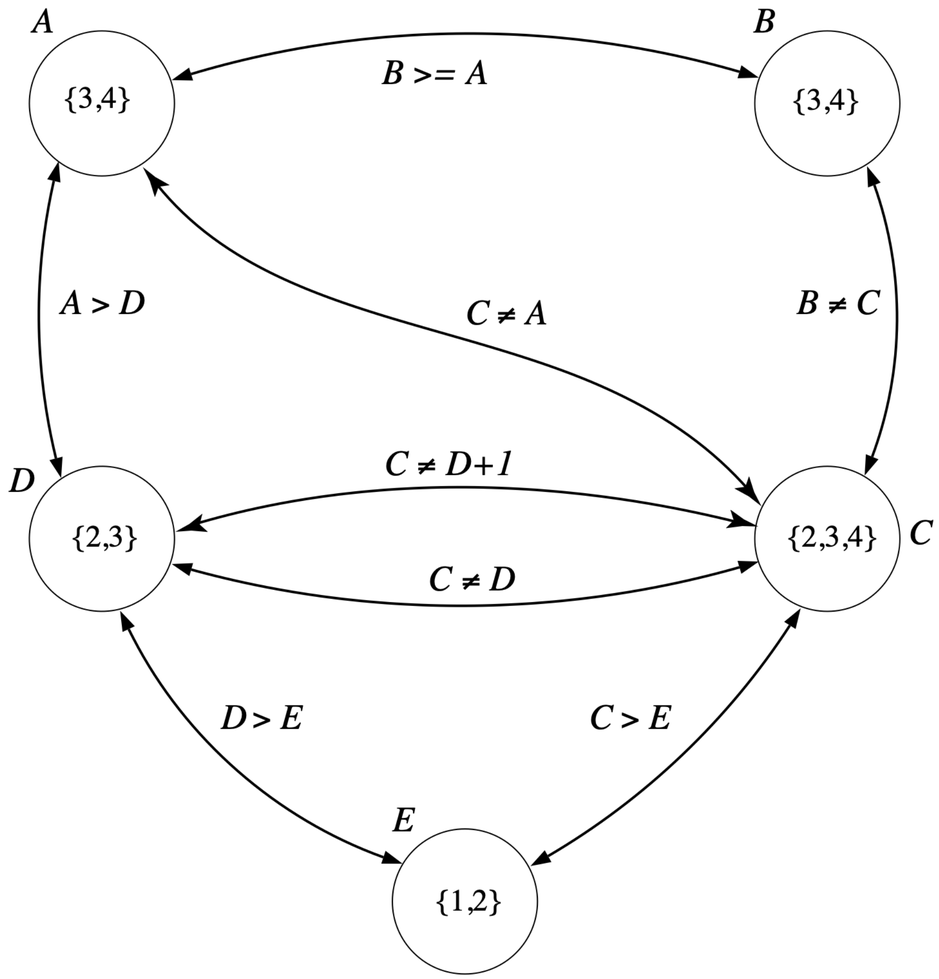
AC exercise

Arc	Relation	Values Removed
<D, E>	D > E	D = 1
<E, D>	D > E	E = 4
<C, E>	C > E	C = 1
<D, A>	A > D	D = 4
<A, D>	A > D	A = 1 & A= 2
<B, A>	B >= A	B = 1 & B = 2
<E, D>	D > E	E = 3



Arc Consistency

Arc	Relation	Value(s) Removed
$\langle D, E \rangle$	$D > E$	$D = 1$
$\langle E, D \rangle$	$D > E$	$E = 4$
$\langle C, E \rangle$	$C > E$	$C = 1$
$\langle D, A \rangle$	$A > D$	$D = 4$
$\langle A, D \rangle$	$A > D$	$A = 1 \& A = 2$
$\langle B, A \rangle$	$B \geq A$	$B = 1 \& B = 2$
$\langle E, D \rangle$	$D > E$	$E = 3$



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

Your feedbacks

- Assessment Schedules for both quizzes and assignments are now released
- Two consultations are hybrid, and one is online
- I try to avoid using handwriting camera
- Please remind me not to use physical laser
- I try my best to repeat questions on the microphone
- If any part needs further explanation, please let us know

Constraint Ordering is Important

```
solve(A, B, C, D, E) :-
    domain(C),
    domain(D),
    domain(A),
    domain(B),
    domain(E),
    A > D,
    D > E,
    C =\= A,
    C > E,
    C =\= D,
    B >= A,
    B =\= C,
    C =\= D + 1.
```

Much faster !

```
solve(A, B, C, D, E) :-
    domain(C),
    domain(D),
    C =\= D,
    C =\= D + 1,
    domain(A),
    A > D,
    C =\= A,
    domain(B),
    B >= A,
    B =\= C,
    domain(E),
    C > E,
    D > E.
```

```
domain(1).
domain(2).
domain(3).
domain(4).
```