University of
South Australia

COMP 2019

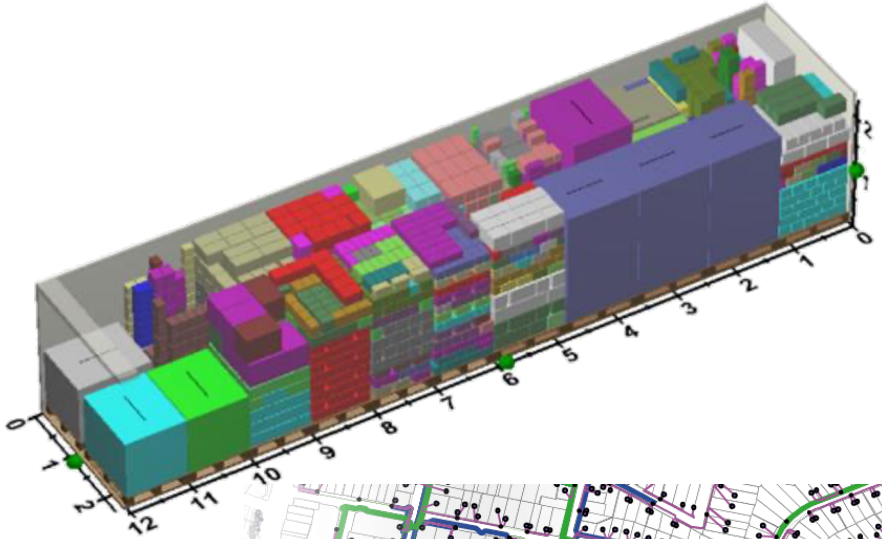
Week 4

Constraint Satisfaction Search

Learning Objectives

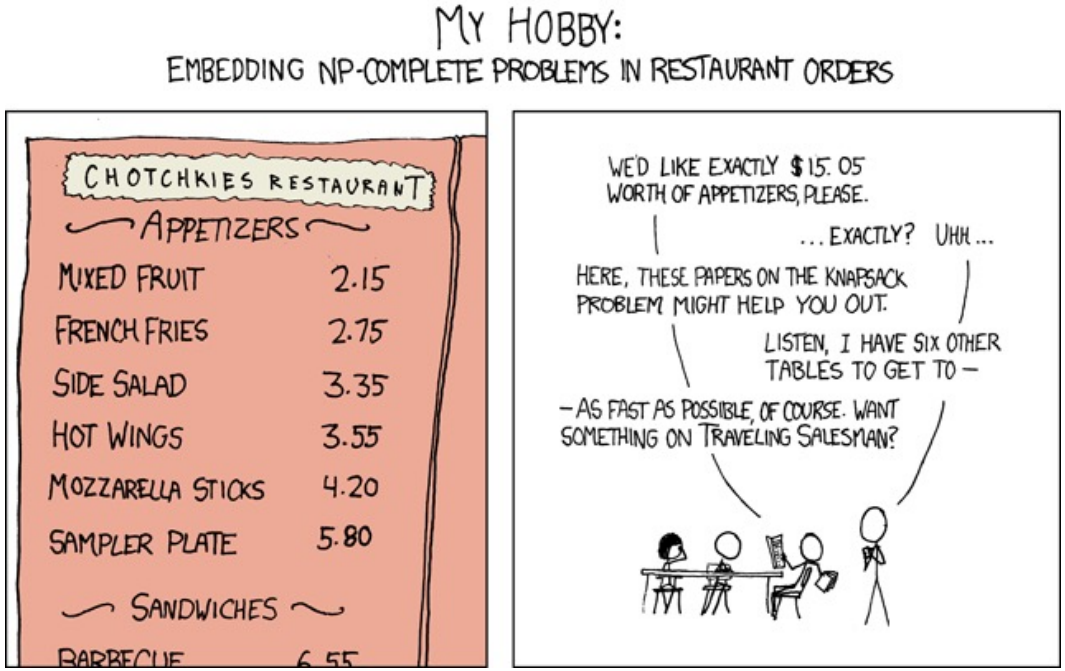
- Explain how constraint satisfaction problems are represented (CO2)
- Explain heuristic methods for Constraint Satisfaction Search (CO2)

Logistics and Optimization



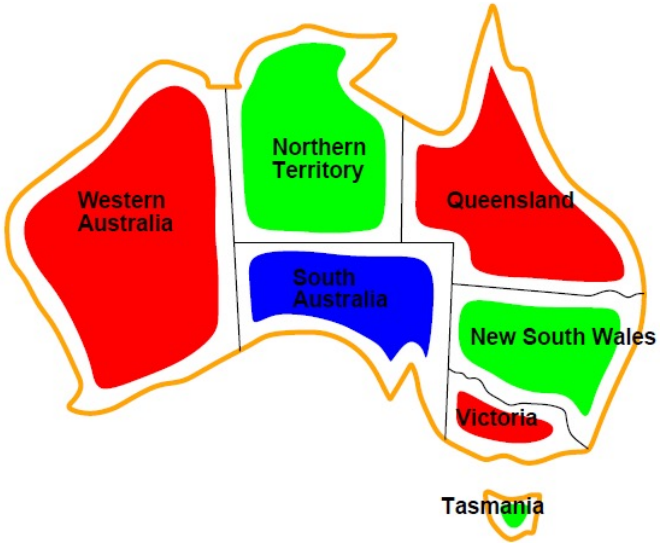
http://www.ortec.com/us/Solution/Pallet_and_Load_Building.aspx http://biarrinetworks.com/network-optimisation

Riddles



http://xkcd.com/287/

Map Colouring Problem



Assign a colour (red or green or blue) to each of the states/territories such that no two adjacent regions are coloured in the same colour?

Constraint Satisfaction Search Problem

- A special kind of search problem
  - Structured state representation that can be solved efficiently
  - Combinatorial optimization problems
- Variables: $\{X_1, \ldots, X_n\}$
- Domains: $D_1, \ldots, D_n$ specify which values each variable may take
- Constraints: $C_1, \ldots, C_k$ restrict the values of some variables
- Solution: Assignment of values to all variables such that all constraints are satisfied

Map Colouring

Variables:

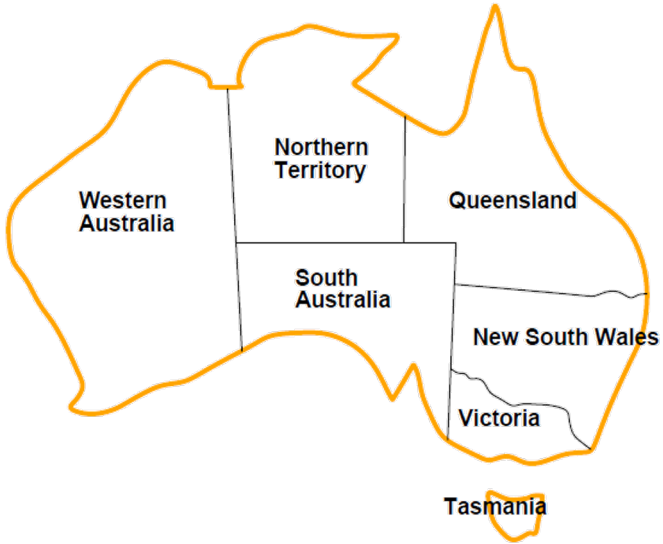
WA, NT, SA, Q, NSW, V, T

Domains:

$D = \{red, green, blue\}$
$(D = D_{WA} = D_{NT} = D_{SA} = D_Q = D_{NSW} = D_V = D_T)$

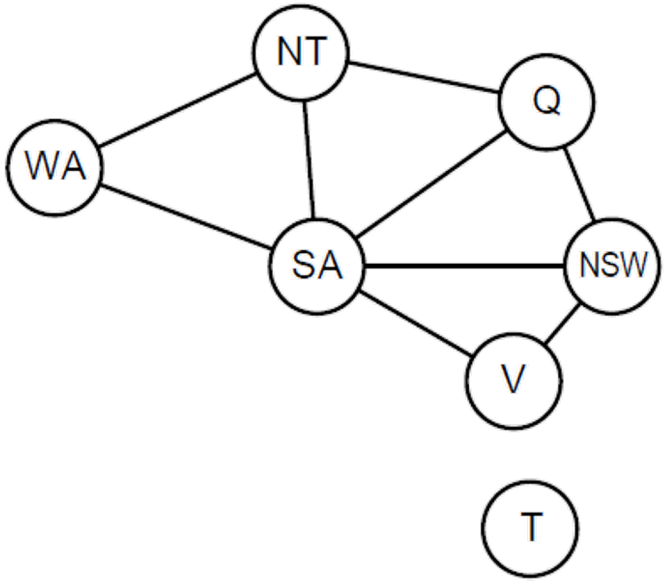
Constraints:

$SA \neq WA, SA \neq NT, SA \neq Q, SA \neq NSW, SA \neq V,$
$WA \neq NT, NT \neq Q, Q \neq NSW, NSW \neq V$



Constraint Graph

Binary CSP: each constraint relates two variables

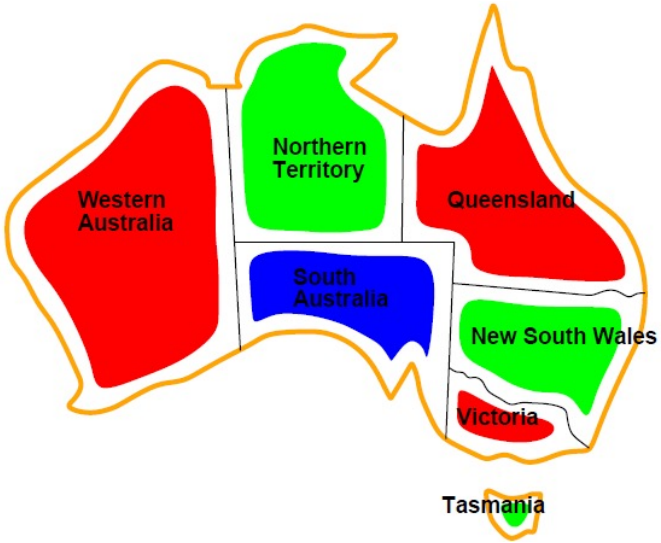


Variables:
WA, NT, SA, Q, NSW, V, T

Constraints:
$SA \neq WA, SA \neq NT, SA \neq Q, SA \neq NSW, SA \neq V,$
$WA \neq NT, NT \neq Q, Q \neq NSW, NSW \neq V$

Solution

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



Assignment of values to all variables such that all constraints are satisfied

Solving CSPs using ”Traditional” Search

- **States** are defined by values assigned so far
- **Initial state**: empty assignment
- **Goal test**: the current assignment is complete
- **Actions**: Assign a value to an unassigned variable that does not conflict with the current assignment
- Every candidate solution appears at depth n (n=number of variables)
  - How can we search efficiently?

Naïve Generate and Test

```
for (WA,NT,SA,Q,NSW,V,T) in
                    [(Red,Red,Red,Red,Red,Red,Red),
                     (Red,Red,Red,Red,Red,Red,Green),
                     (Red,Red,Red,Red,Red,Red,Blue),
                       ….
                     (Blue,Blue,Blue,Blue,Blue,Blue,Blue)]:
          if IsGoal(WA,NT,SA,Q,NSW,V,T):
                    return (WA,NT,SA,Q,NSW,V,T)
return No_Solution
```

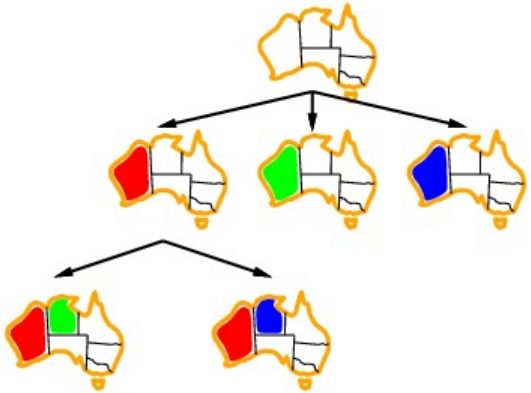
Backtracking Search to Enumerate Candidates

- Variable assignments are commutative
  - WA = red, NT = green
    is the same as
    NT = green, WA = red.
- Sufficient to consider a single variable for each action
- DFS with single-variable assignments
- Extend assignments until solution or inconsistency

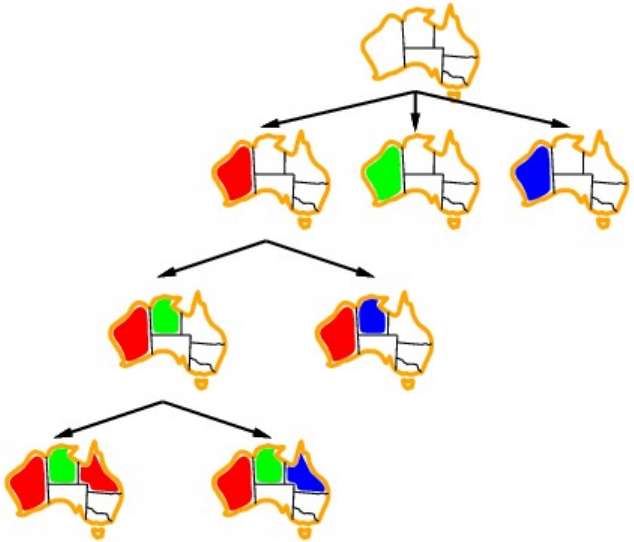
Backtracking Search

Backtracking Search

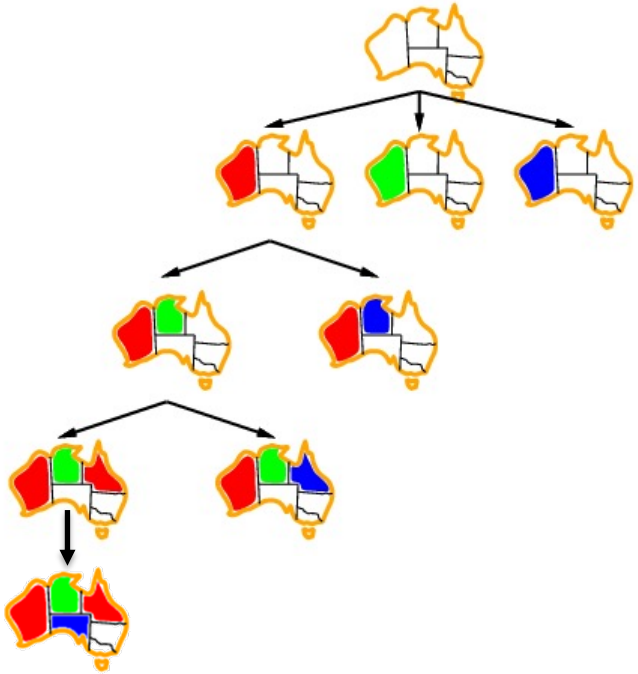
Backtracking Search



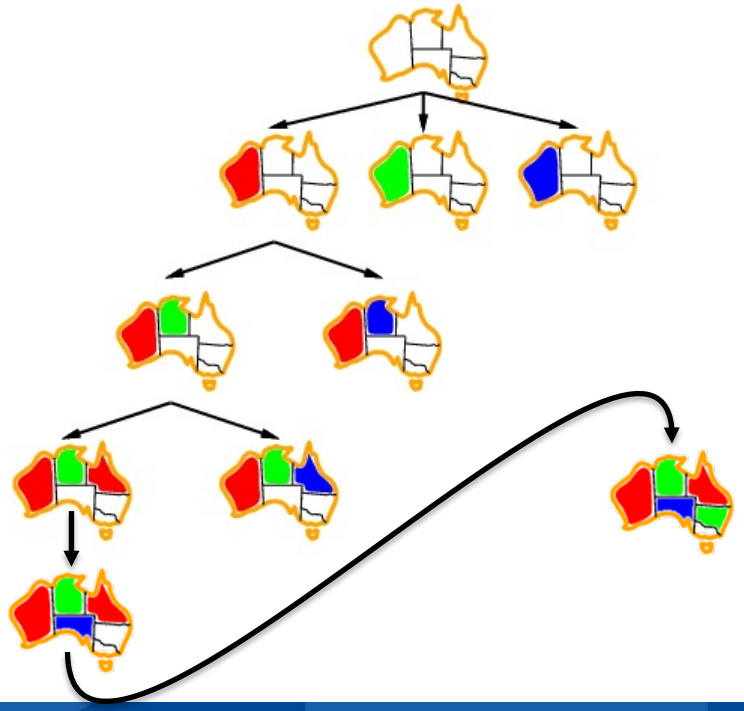
Backtracking Search



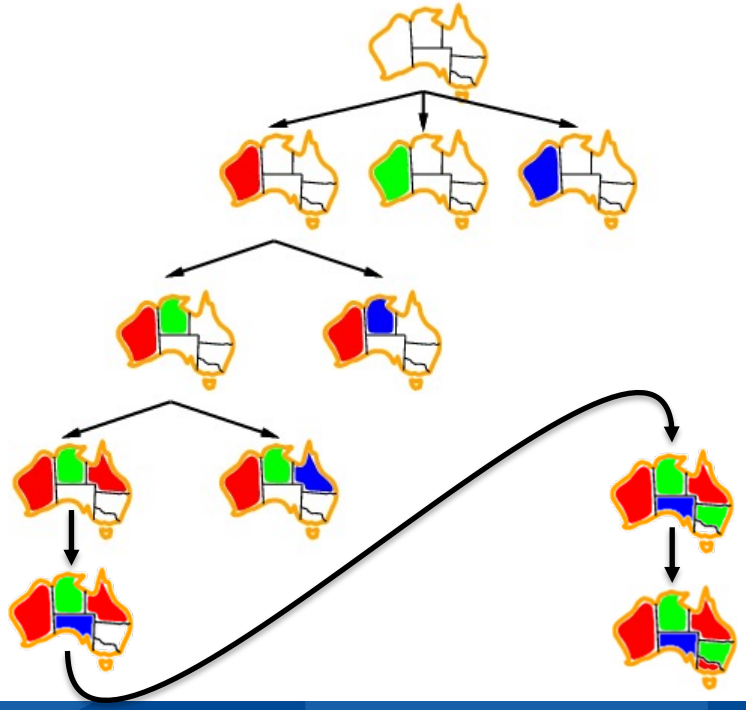
Backtracking Search



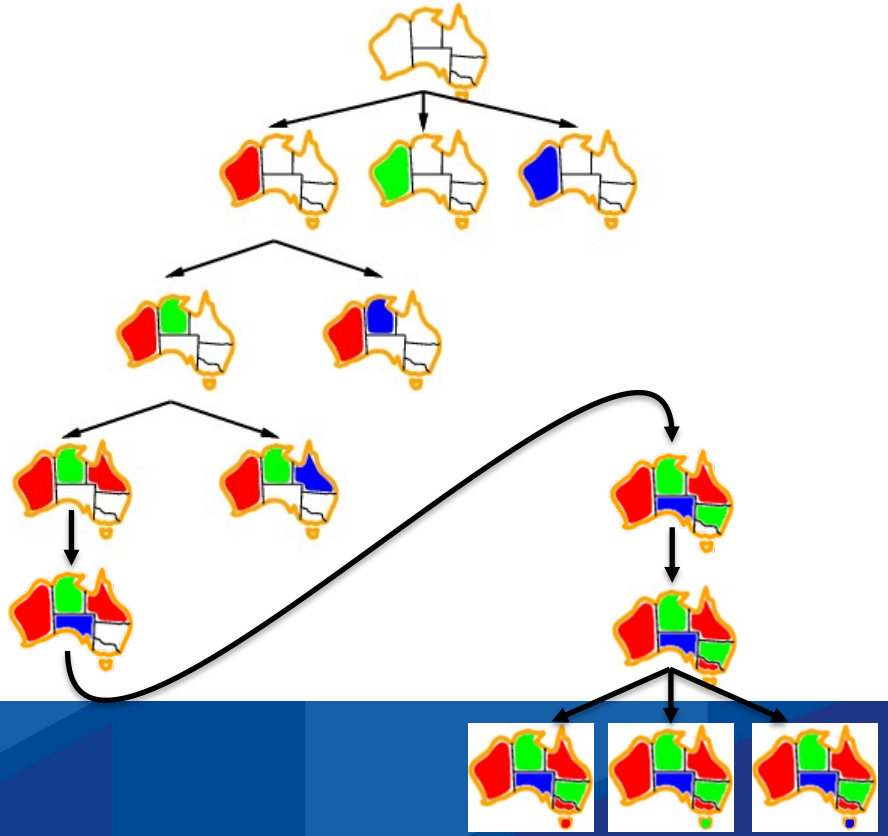
Backtracking Search



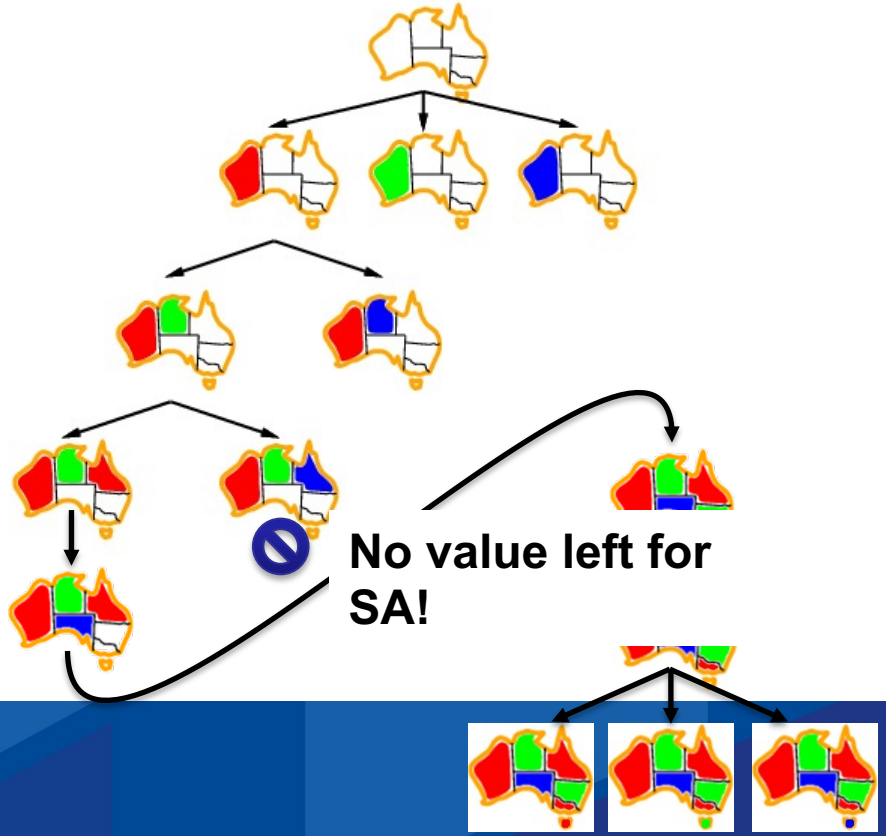
Backtracking Search



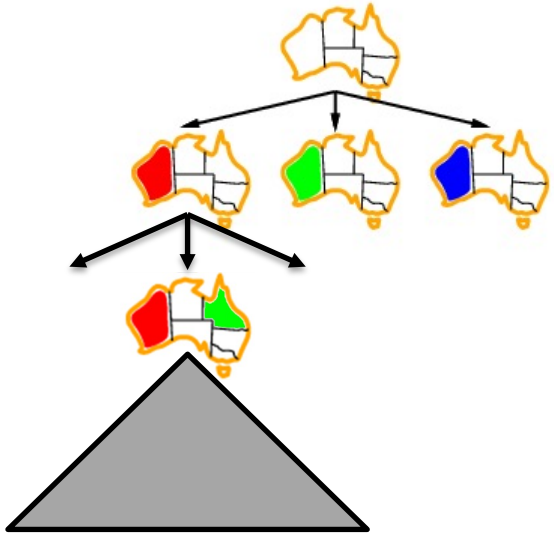
Backtracking Search



Backtracking Search

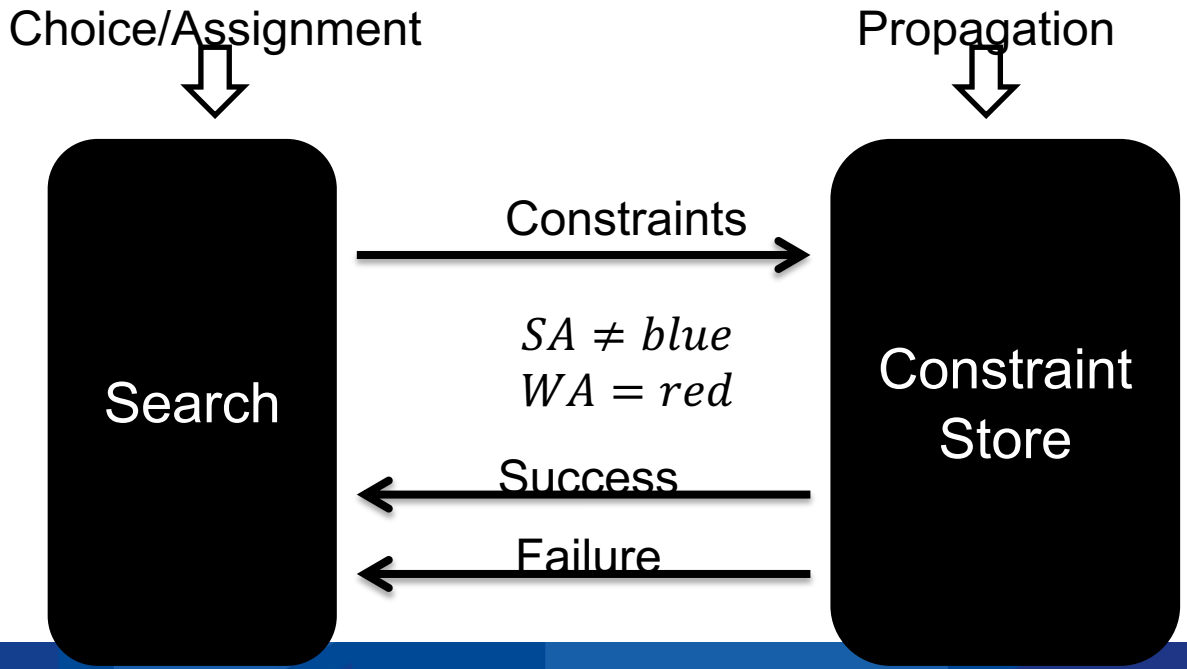


Backtracking Search



No solution in here!

Computational Paradigm



Role of Constraints

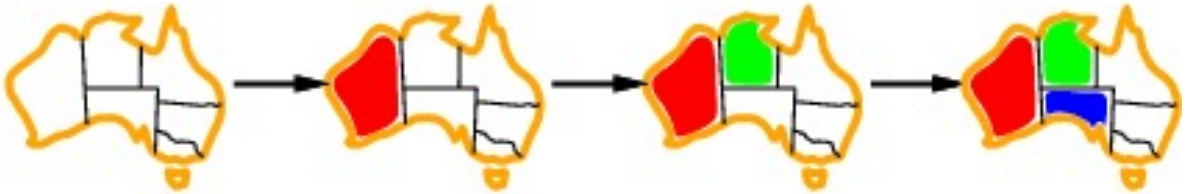
- Feasibility checking
  - Does the current assignment of values to variables satisfy the requirements of a valid solution?
- Pruning
  - Determine which values cannot be part of any solution that extends the current assignment

General Heuristics are Efficient

- General-purpose methods can yield huge gains in speed
  - Which variable should be assigned next?
  - In what order should its values be tried?
  - Can we detect inevitable failure early?

Most Constrained Variable Heuristic

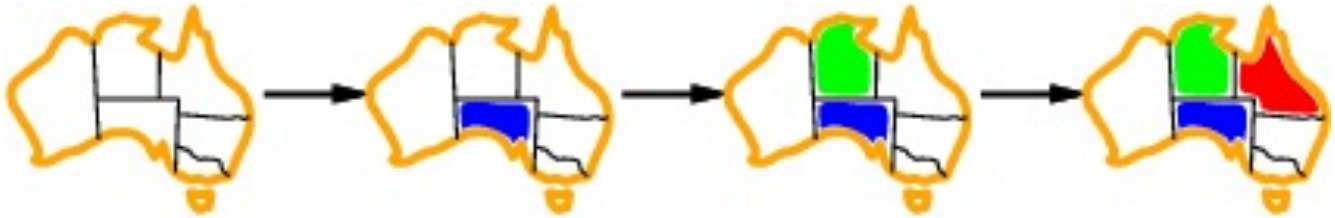
- Choose the variable with the fewest legal values
  - Focus on “difficult” variables which may prevent a solution being found



- minimum remaining values (MRV) heuristic

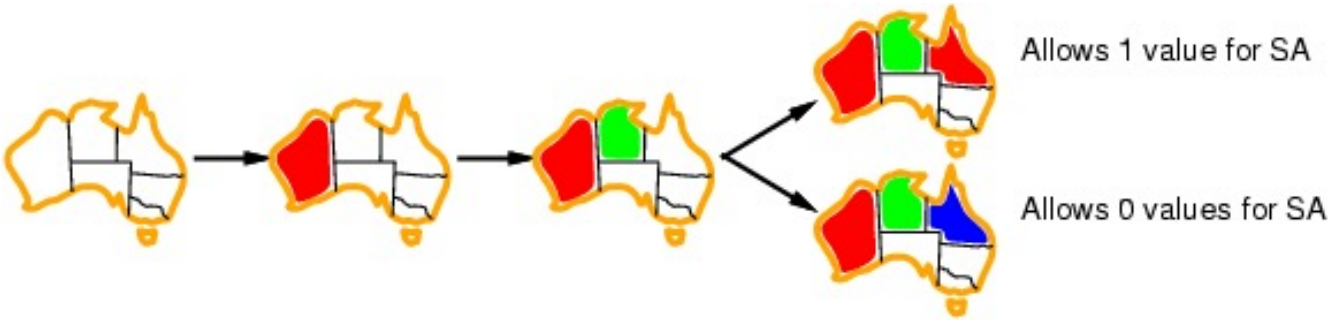
Most Constraining Variable

- Choose the variable with the most constraints
  - Tie-breaker among most constrained variables



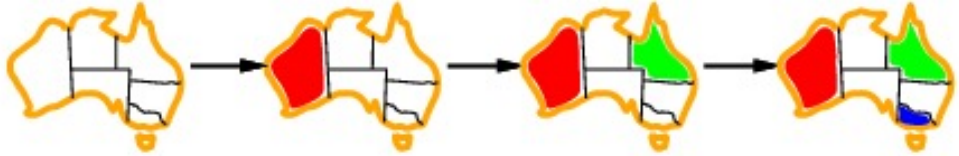
Least Constraining Value

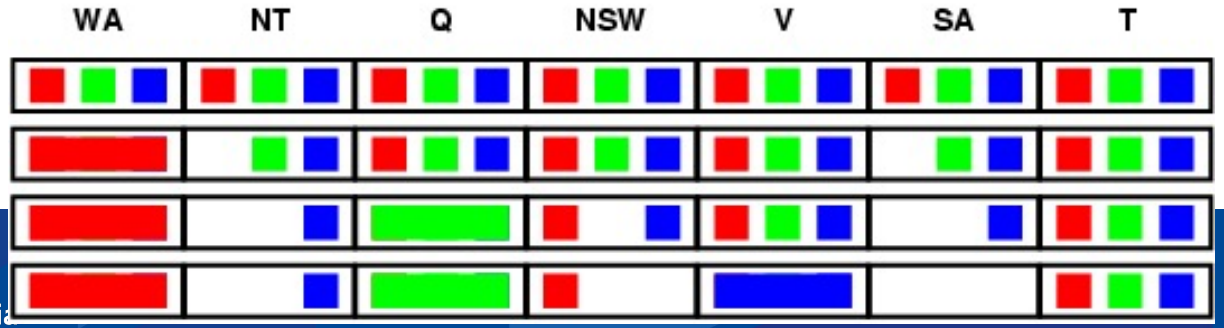
- Choose the least constraining value for a given variable:
  - rules out the fewest values in the remaining variables



Forward Checking

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

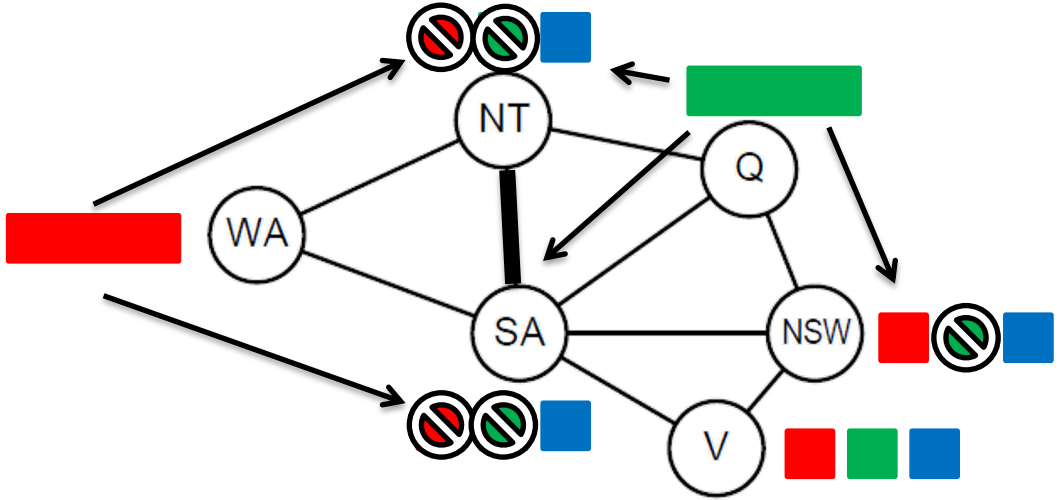




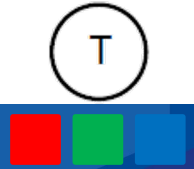
Forward Checking is not Enough

- Forward Checking does not detect all failures

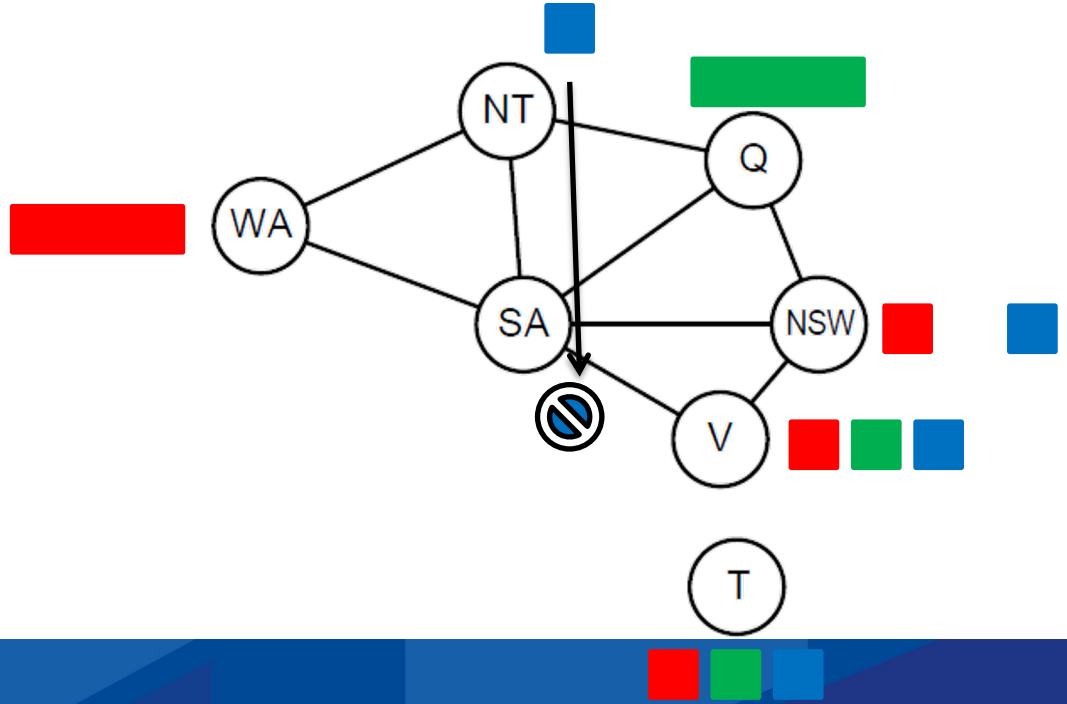
Forward Checking is not Enough



$NT \neq SA$ is not detected

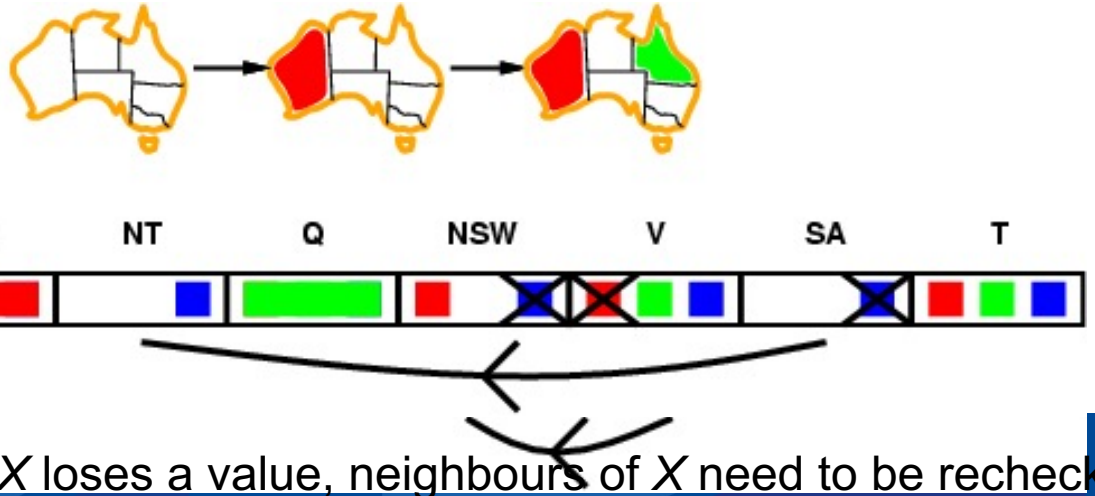


Forward Checking is not Enough



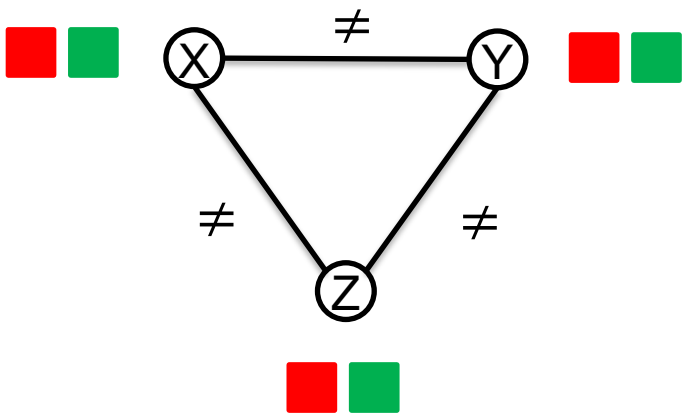
Arc consistency

- Makes each arc consistent
  - X →Y is consistent iff
    for every value x of X there is some allowed y of Y



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

Global Consistency



- The graph is arc consistent, but there is no solution
  - Arc consistency does not imply global consistency
  - Can be used as filtering mechanism

Global Constraints

- Binary constraints are inefficient in some domains
- "Global" constraints relate many variables
  - "All different", "Sum", "Max", "Bipartite Matching", etc
  - Specialized, efficient propagation algorithms

$X_{11}, \ldots, X_{99} \in \{1, \ldots, 9\}$

all_different($X_{11}, X_{12}, X_{13}, X_{21}, X_{22}, X_{23}, X_{31}, X_{32}, X_{33}$)

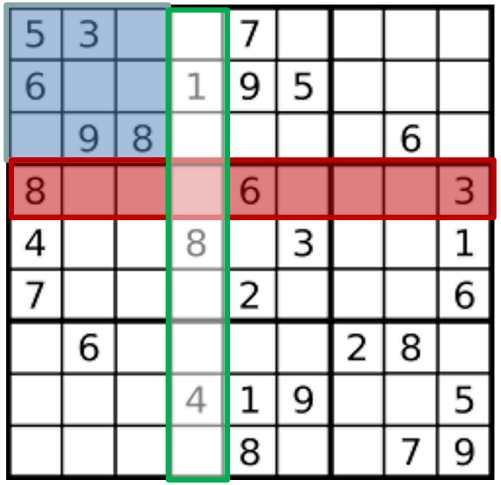
…

all_different($X_{41}, X_{42}, \ldots, X_{49}$)

…

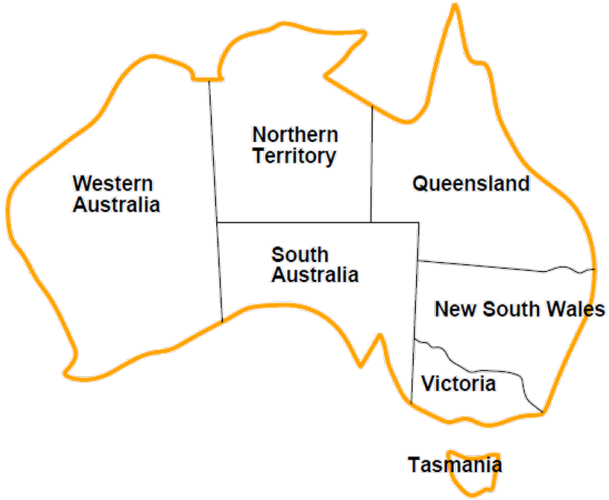
all_different($X_{14}, X_{24}, \ldots, X_{94}$)

…



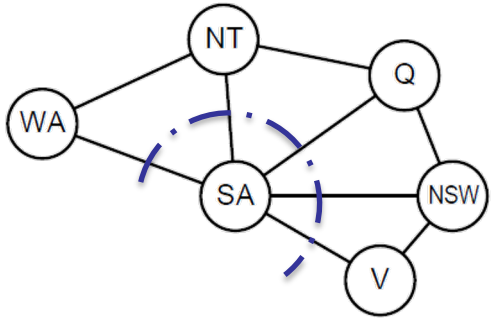
Exploiting the Problem Structure

- Independent sub-problems
  - can be solved separately
  - Solutions for the entire problem can be found by composing the sub-problems' solutions.



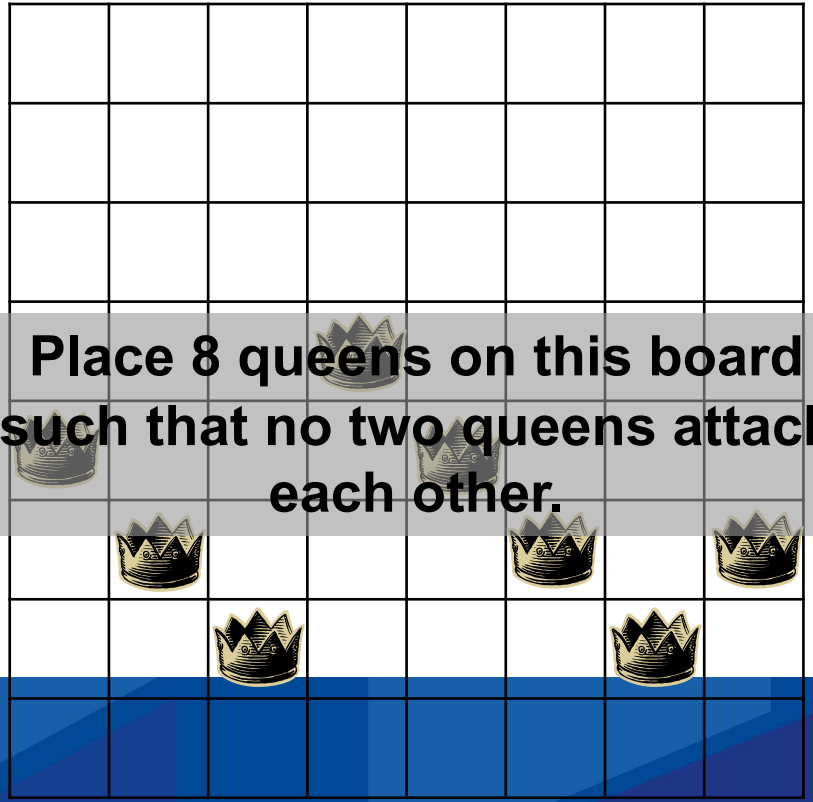
Exploiting the Problem Structure

- Tree structures CSPs can be solved very quickly without backtracking
- Cut set conditioning: Graphs that are nearly trees can be solved by assigning the variables not in the spanning tree first.



If it has no loops, it can be solved in $O(n \times d^2)$

Local Search



Place 8 queens on this board such that no two queens attack each other.

Local Search

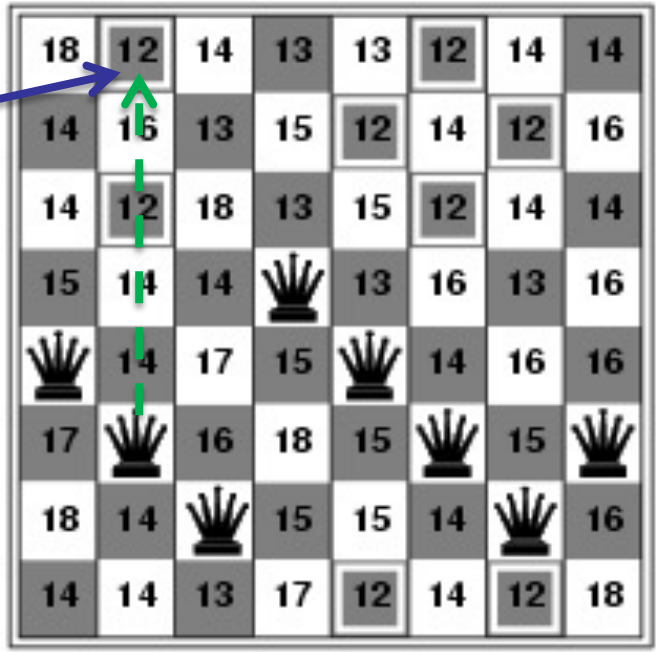
- Start with a complete assignment
  - Try to repair/improve the current assignment by making small changes
  - Minimisation of “cost” associated with constraint violations => **Optimisation**
- Repeat:
  - Make changes to “repair” constraint violations
  - Stop if all constraints are satisfied or no improvement

5 violations 2 violations 0 violations

Local Search Heuristics

- Variable selection:
  - Max-conflict heuristic: choose variable that appears in the most violated constraints
  - Randomly select any conflicted variable
- Value selection:
  - Min-conflicts heuristic: choose value that violates the fewest constraints
- Actions
  - Reassign values to variables
  - Swap values of variables
  - …
- Many powerful problem-specific heuristics exist

Total violations
if queen in column 2
is moved here



Violations/Queen: 2 **5** 4 4 **5** **5** **5** 3

Summary

- Constraint satisfaction search is an efficient method for solving many industrial problems, such as scheduling, allocation, configuration
- Domain knowledge is expressed as constraints
- General heuristics are at the heart of efficient search
- There are off the shelf tools for solving CSPs
- Local repair methods can be applied to problems that are too large for exhaustive search

University of
South Australia

Questions?