Constraint Satisfaction, Part 1

Lecture 11

Chapter 6, Sections 6.1-6.3

Jim Rehg

College of Computing

Georgia Tech

February 8, 2016

Administrative Updates

Project 1 is due Sunday Feb 14 at midnight

The submission site closes at 1am on Feb 15

Be sure to upload your solutions before the site closes

You have a one hour grace period in case of network issues etc.

No late submissions will be accepted

Midterm exam will be held in-class on Monday Feb 29

Exercises on combinatorial search (Chapter 3) released today

These will not be graded, but we will release solutions

Search So Far

Classical (combinatorial) search:

State space has no *explicitly represented* structure

Implicit structure in generating successors and testing for goals

Heuristics encode *problem-specific* information to guide search

Constraint Satisfaction

Type of search problem, goal is to *assign values to variables*

Structure of state space and goal test can be exploited

Develop *general-purpose* reusable solution methods

Constraint Satisfaction Problem: Definition

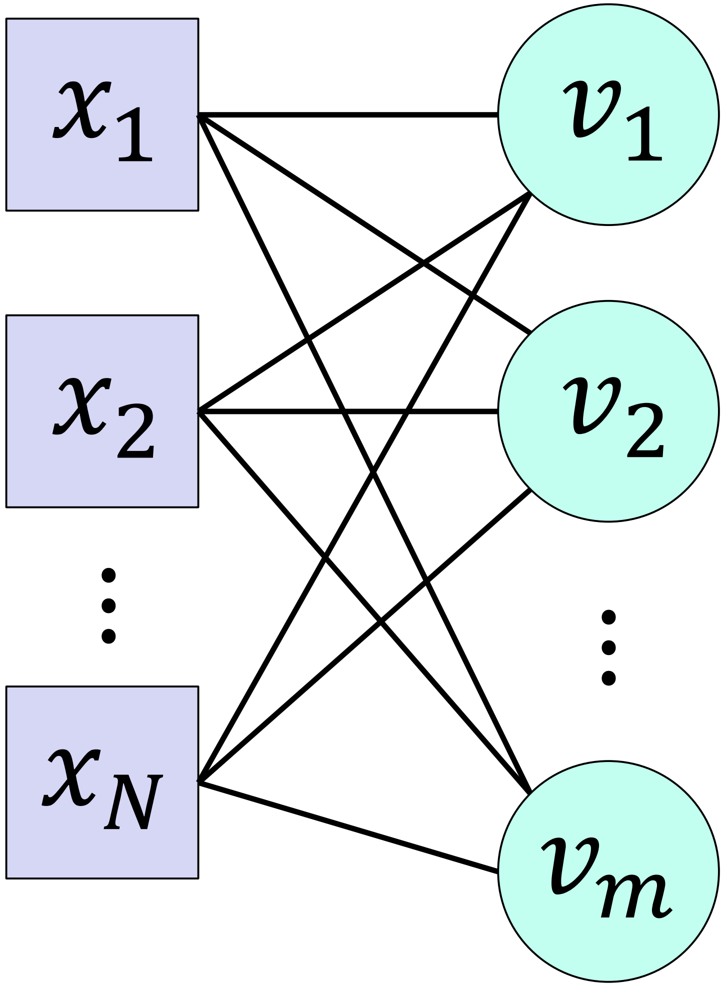
State Representation

Set of Variables: $\{x_1, \ldots, x_N\}$

Domain of Values: $\{v_1, \ldots, v_m\}$

Goal Test

Constraints specify allowed assignments



Example of formal representation language

Enables useful general purpose methods

Constraint Satisfaction Problem: Definition

State Representation

Set of Variables: $\{x_1, ..., x_N\}$

Domain of Values: $\{v_1, ..., v_m\}$

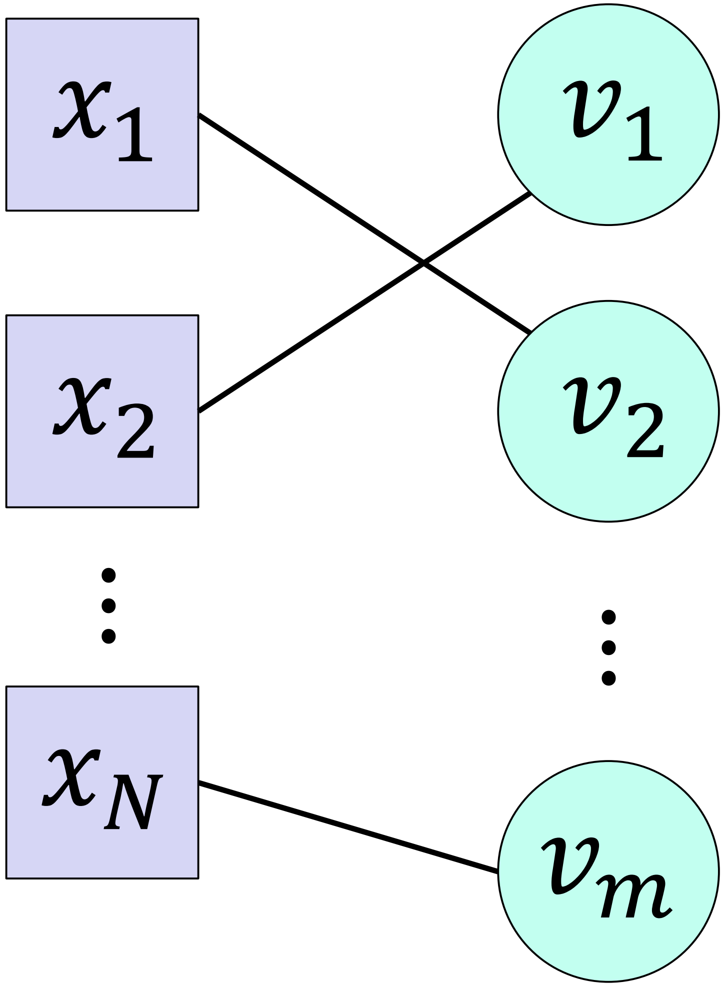
Goal Test

Constraints specify allowed assignments

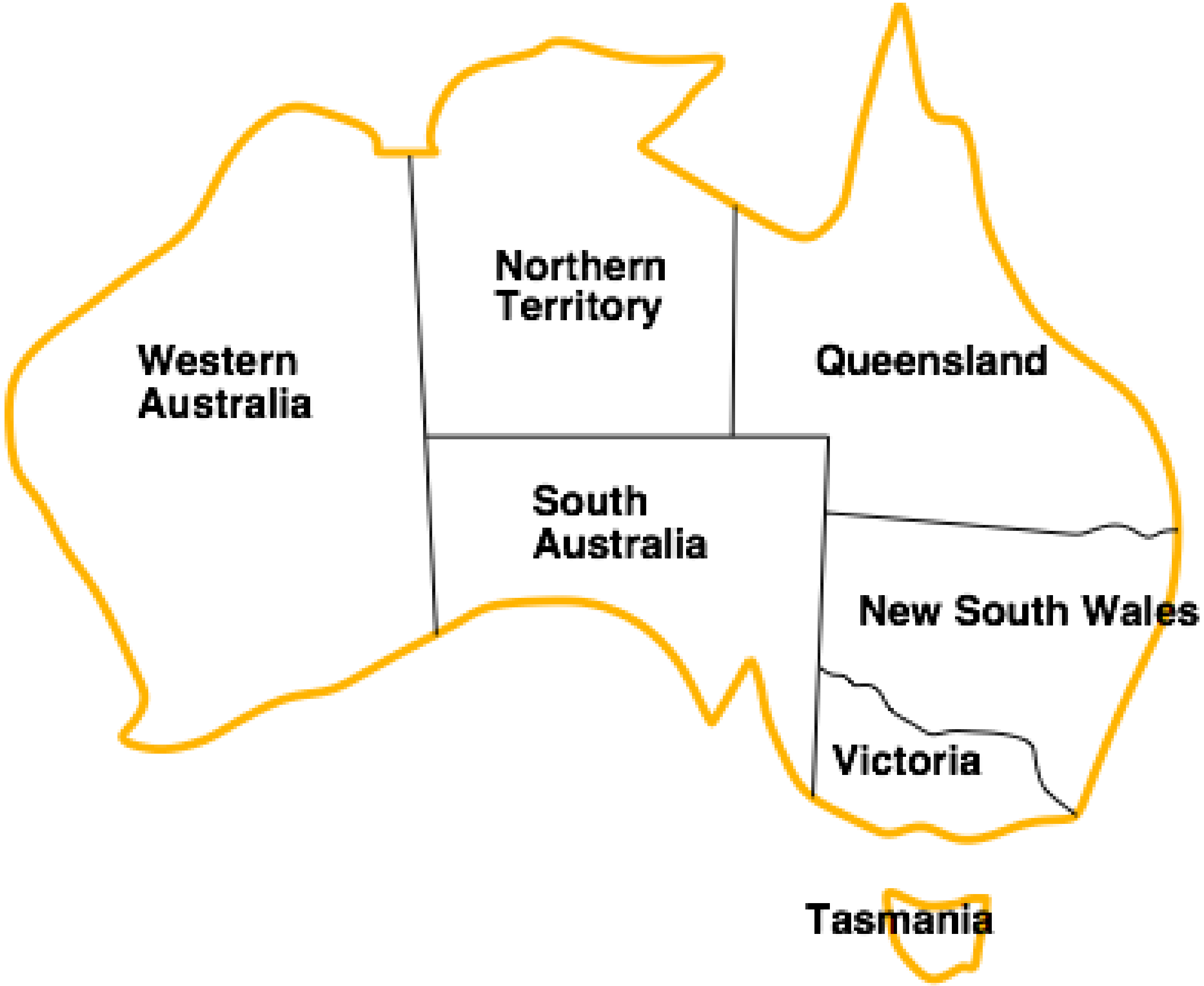
Example: ALL-DIFF

Each variable is assigned a value

No value is assigned more than once



Example: Map Coloring



Variables

- WA, NT, Q, NSW, V, SA

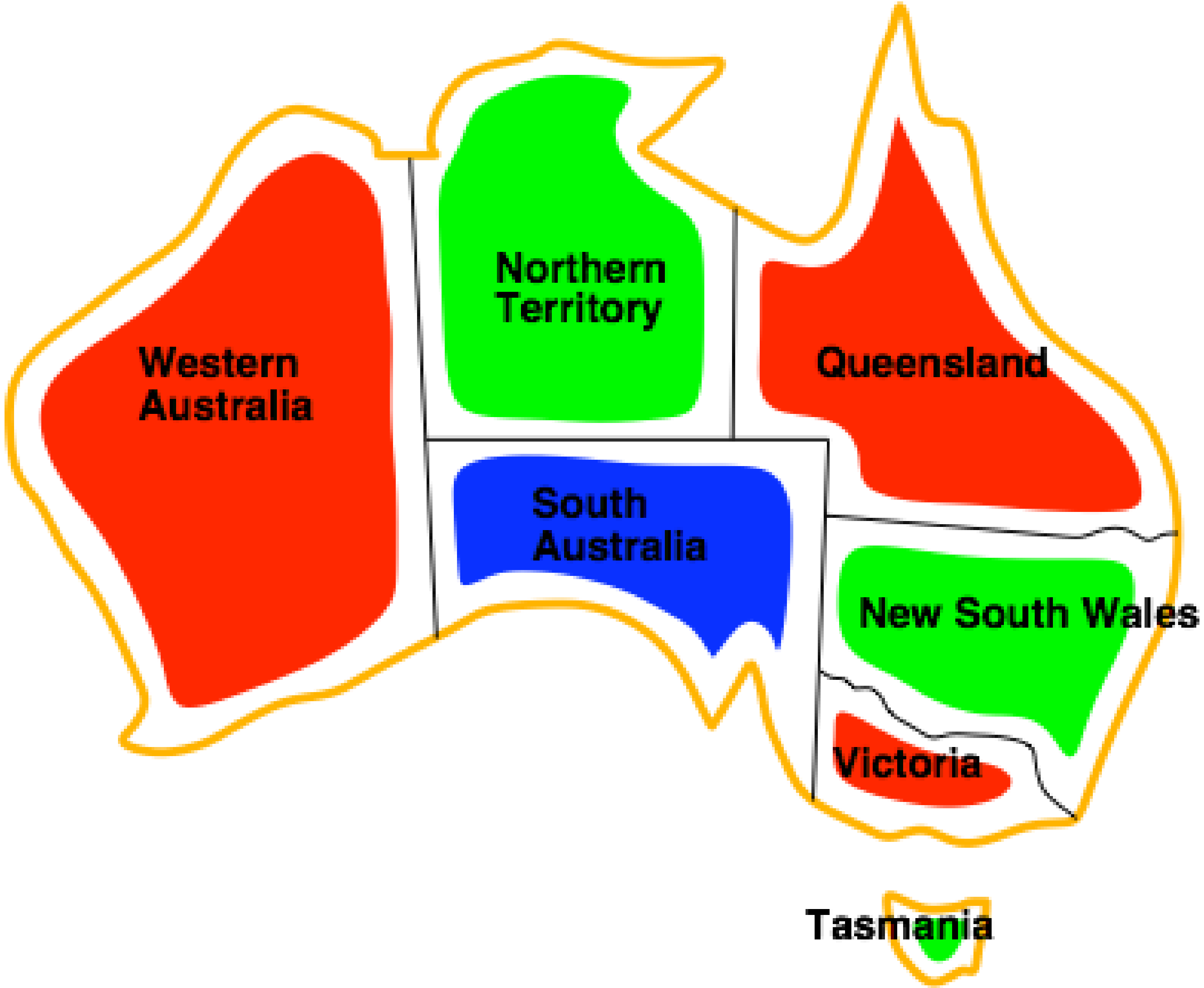
Values

- {Red, Green, Blue}

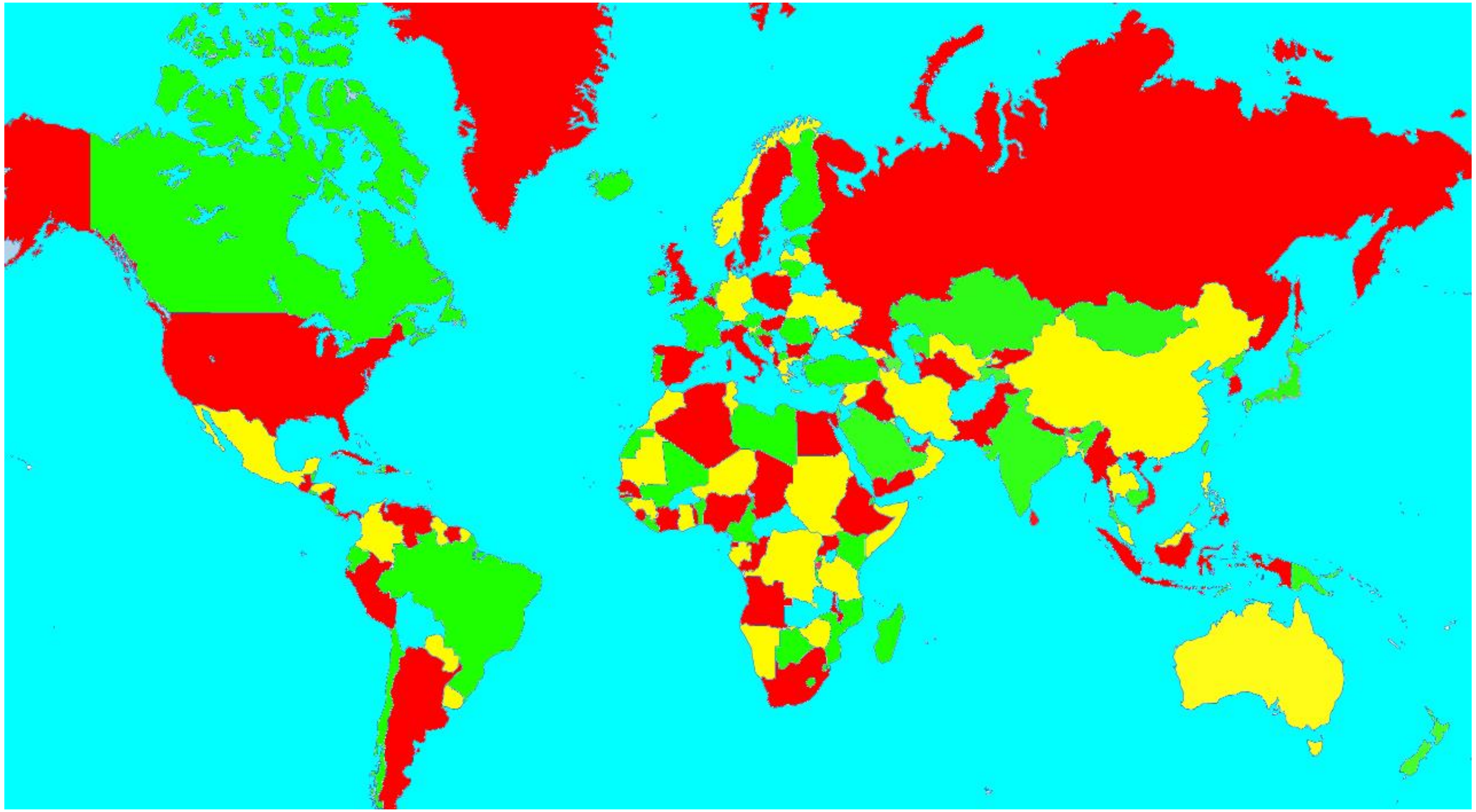
Constraints

- Adjacent regions must have different colors
- e.g. $WA \neq NT$
- e.g. $(WA, NT) \in \{(R, G), (R, B), (G, R), (G, B), \ldots\}$

Map Coloring Solution

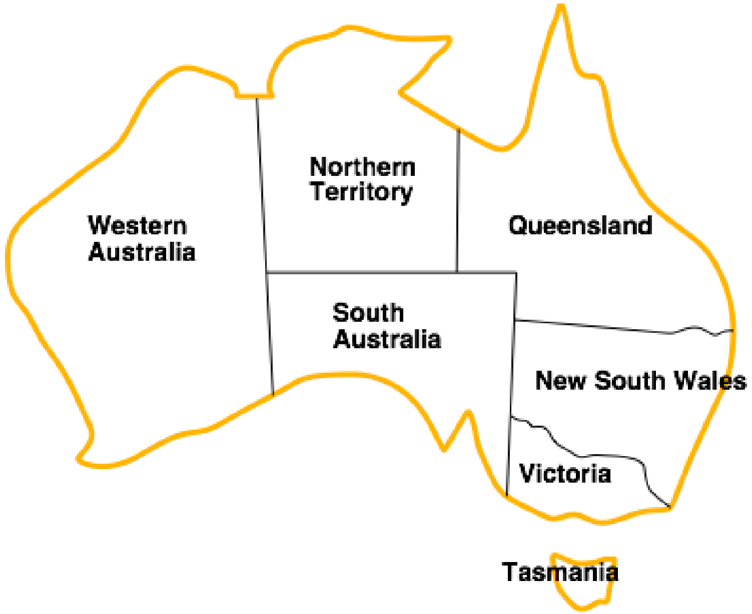


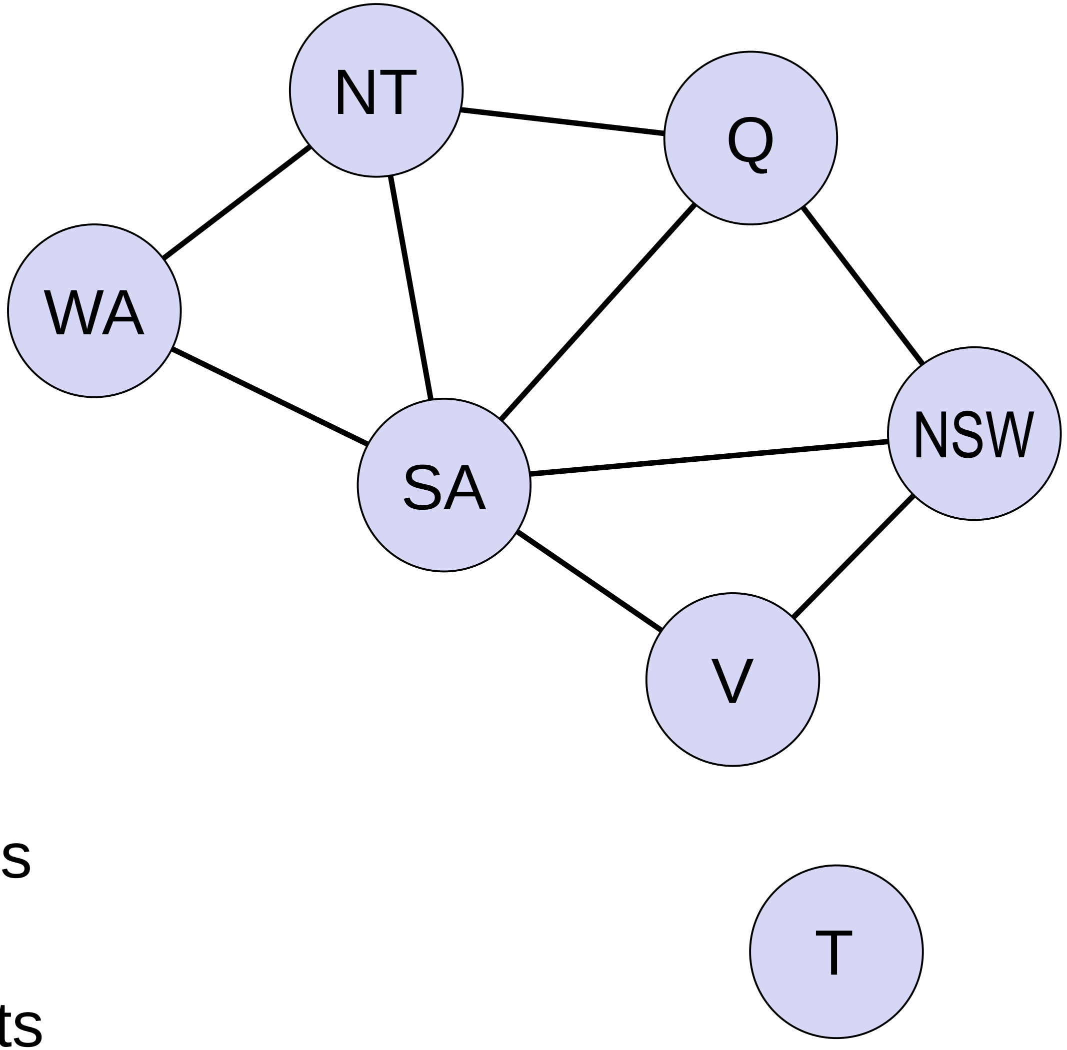
4 Color Theorem



Only four colors are needed to color a planar map

Constraint Graph Representation





Nodes are variables

Arcs connect constrained variables

Binary CSP – Each constraint relates only two variables

In this example, inequality constraints

Example: Car Pool Scheduling

Schedule 4 people into 2 cars,

satisfying constraints

Variables = $\{P_1, P_2, P_3, P_4\}$

Values = $\{C_1, C_2\}$

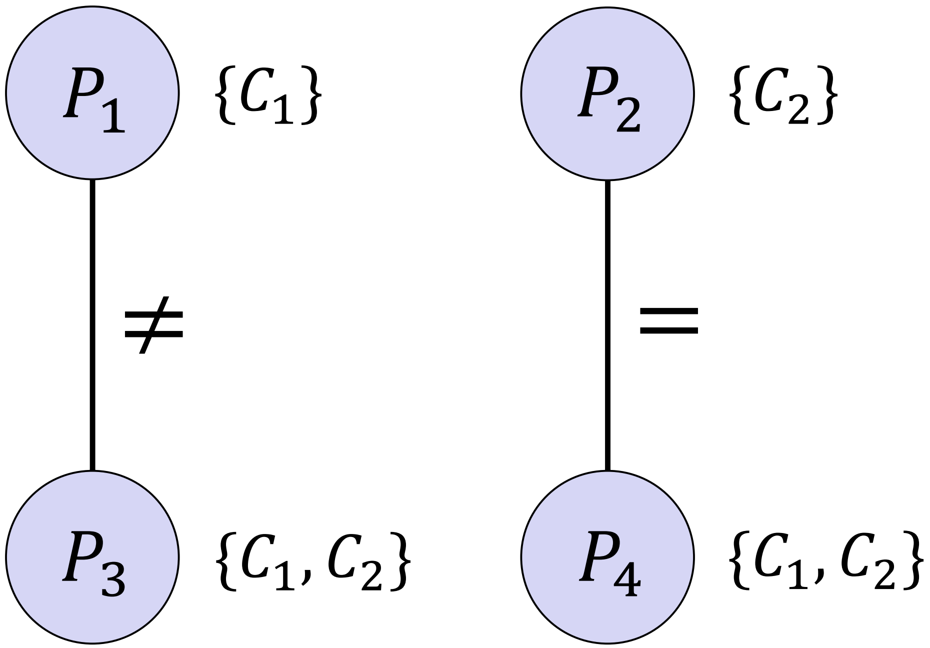
Constraints

$P_1 = C_1$

$P_2 = C_2$

$P_1 \neq P_3$

$P_2 = P_4$



Possible Solution: $P_1 = C_1$, $P_2 = C_2$, $P_3 = C_2$, $P_4 = C_2$

Example: Class Scheduling

Schedule 2 classes for 2 profs into 3 possible time slots

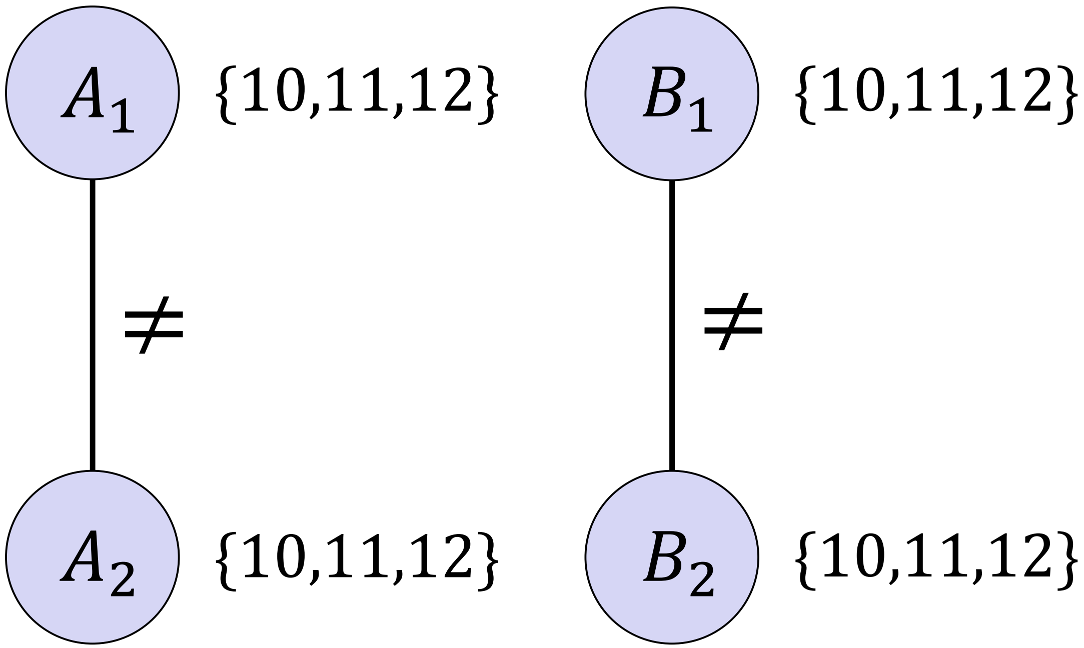
Variables: $\{A_1, A_2, B_1, B_2\}$

Values: $\{10,11,12\}$

Constraints:

$A_1 \neq A_2$

$B_1 \neq B_2$



Possible Solution: $A_1 = 10$, $A_2 = 11$, $B_1 = 11$, $B_2 = 12$

Example: Class Scheduling

Schedule 2 classes for 2 profs into 3 possible time slots

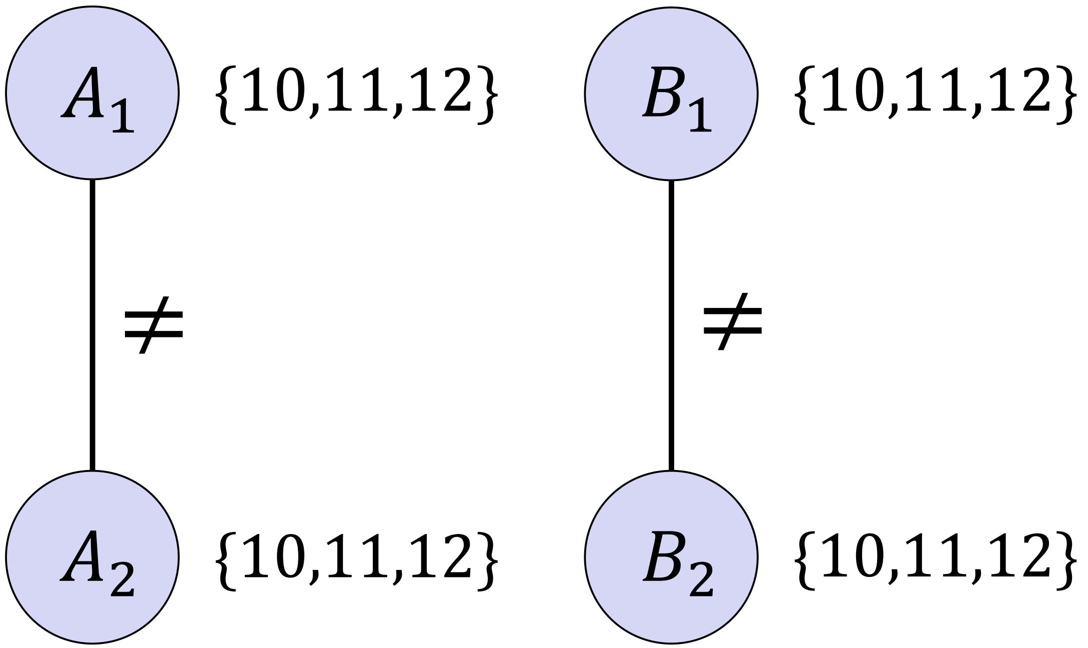
Variables: $\{A_1, A_2, B_1, B_2\}$

Values: $\{10,11,12\}$

Constraints:

$A_1 \neq A_2$

$B_1 \neq B_2$



Now suppose that only one classroom is available.
How could you capture this additional constraint on the assignments?

Example: Class Scheduling

Schedule 2 classes for 2 profs into 3 possible time slots

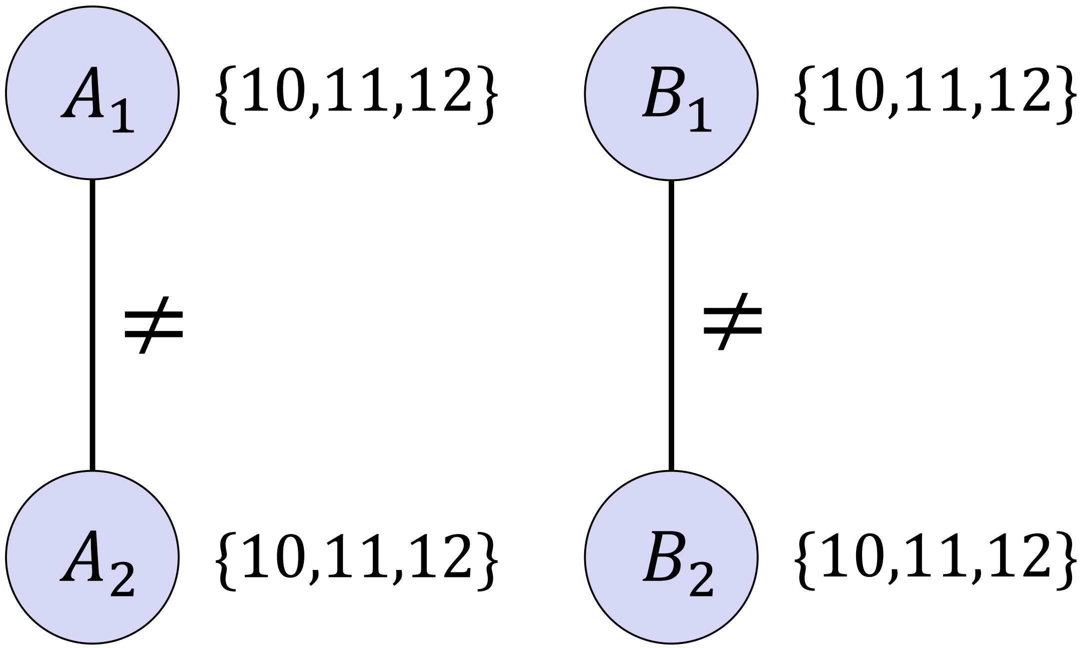
Variables: $\{A_1, A_2, B_1, B_2\}$

Values: $\{10,11,12\}$

Constraints:

$A_1 \neq A_2$

$B_1 \neq B_2$



Now suppose that only one classroom is available.
How could you capture this additional constraint on the assignments?
Answer: Create ALL-DIFF problem, no slots can share values
In this case there would be no solution

Example: Sudoku

Populate the squares with numbers from 1-9, satisfying the constraints

Variables: 81 squares

Values: $\{1, \ldots, 9\}$

Constraints:

- Row: All values used exactly once
- Col: All values used exactly once
- Box: All values used exactly once

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

Possible assignments: 9^{81}

Example: N-Queens

Place N Queens on an N x N chessboard such that no Queen can attack another

Variables: N rows

Values: {1,…,N} (Queen position Q_i)

Constraints:

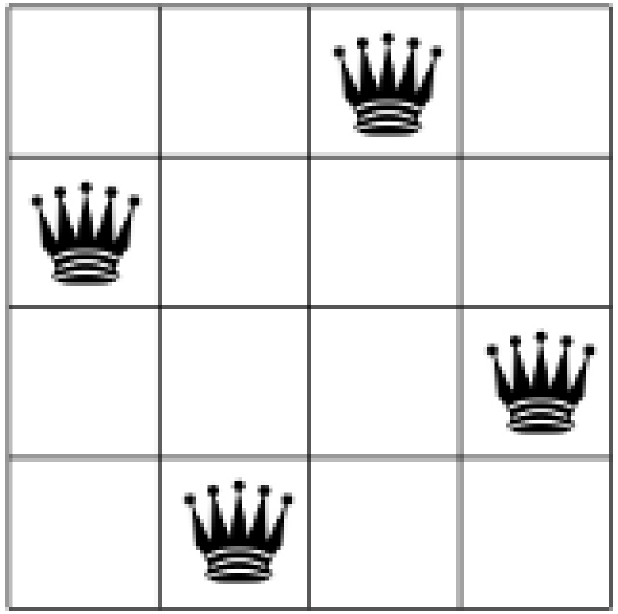
- $Q_i \neq Q_j$ (ALL-DIFF on positions)
- $|Q_i - Q_j| \neq |i - j|$ (different diagonals, also ALL-DIFF)

$Q_1 = 3$

$Q_2 = 1$

$Q_3 = 4$

$Q_4 = 2$



Solution to 4 Queens

Constraint Satisfaction Problems in Discrete Variables

Finite domains

Constraints can be enumerated, $O(m^N)$ complete assignments

Infinite domains

Constraint language needed (e.g. $x_1 + 5 \leq x_2$)

Types of constraints

By arity: unary ($SA \neq$ green), binary ($SA \neq \mathrm{WA}$), higher order (3 or more variables)

Linear constraints:

Assignment problem: Hungarian algorithm (polynomial)

General linear (and nonlinear) constraints: NP

CSPs in Continuous Variables

Example: Job shop scheduling

Start and end times and durations are real-valued

e.g. *Start time for job 2 must occur after job 1 is completed if they require the same resource*.

With start times S_i and durations D_i: $S_1 + D_1 \leq S_2$

Types of constraints

Linear equality constraints: Gaussian elimination (polynomial)

Linear inequality constraints: Linear programming (exponential)

Solving CSPs via Search

State Model

State: assignment to k variables with $k-1, \ldots, N$ unassigned

Legal (consistent) assignment: No constraints violated

Complete assignment: All variables assigned

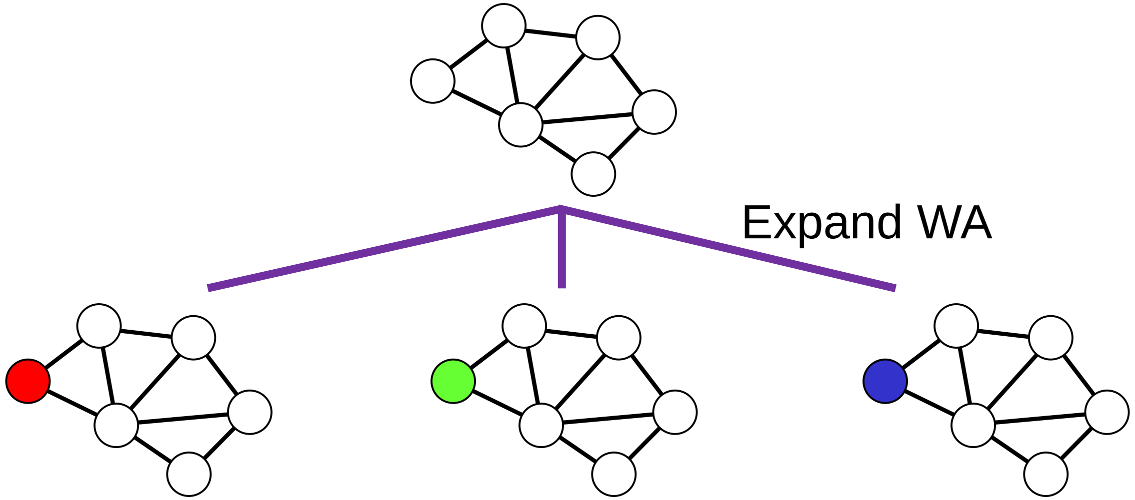
Goal states(s): All complete and consistent assignments

Initial State: All variables are unassigned

Successor State: Assign a value to the next variable (e.g. x_{k+1}), keeping all other variables unchanged

No cost on transitions: just find a solution, don't consider path costs

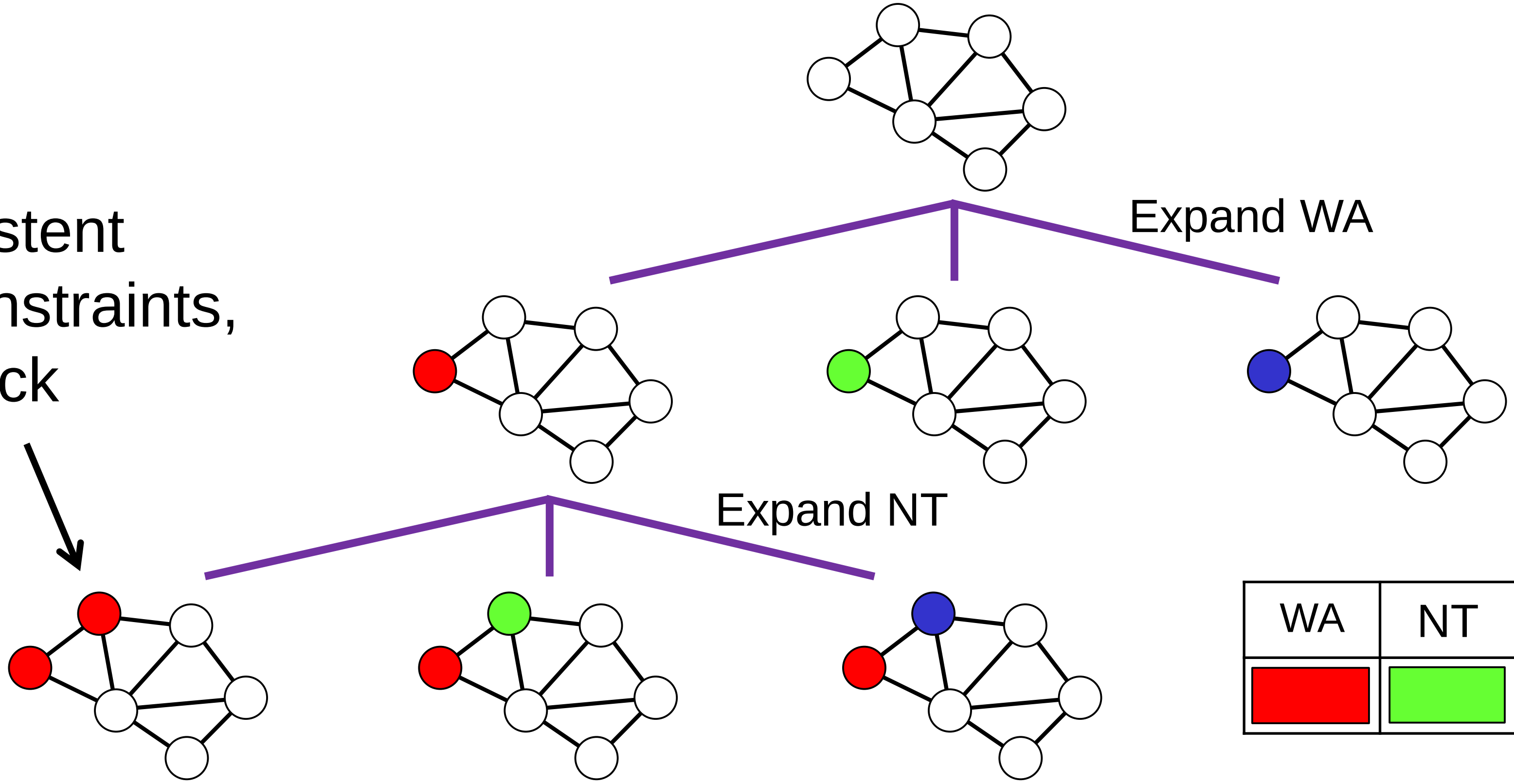
Backtracking Search

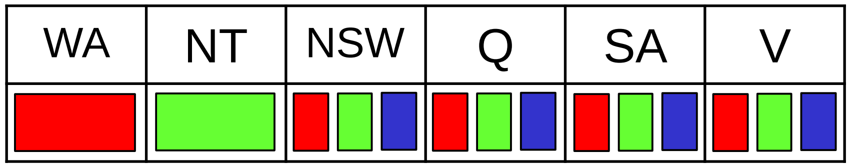


Expand WA

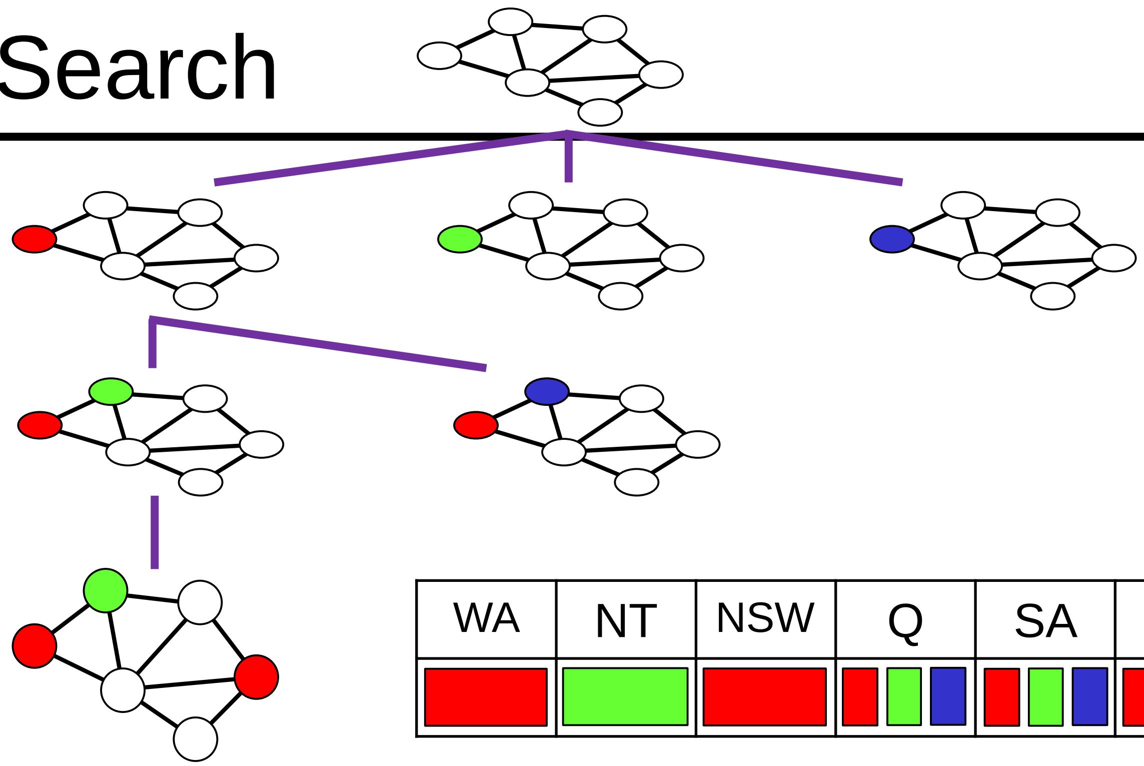
Backtracking Search

Inconsistent with constraints, backtrack

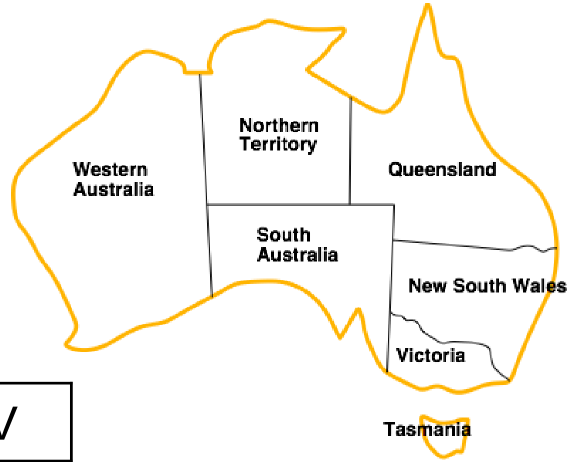




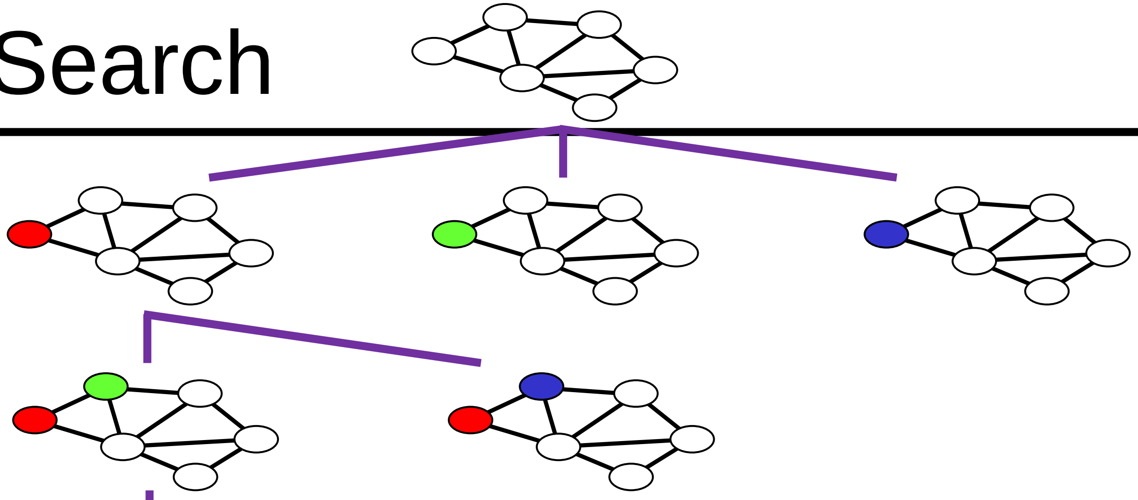
Backtracking Search

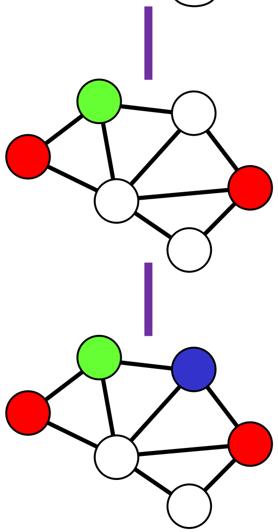


WA	NT	NSW	Q	SA	V



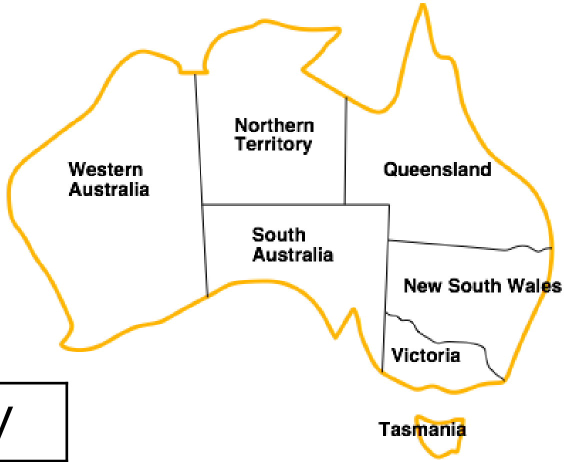
Backtracking Search



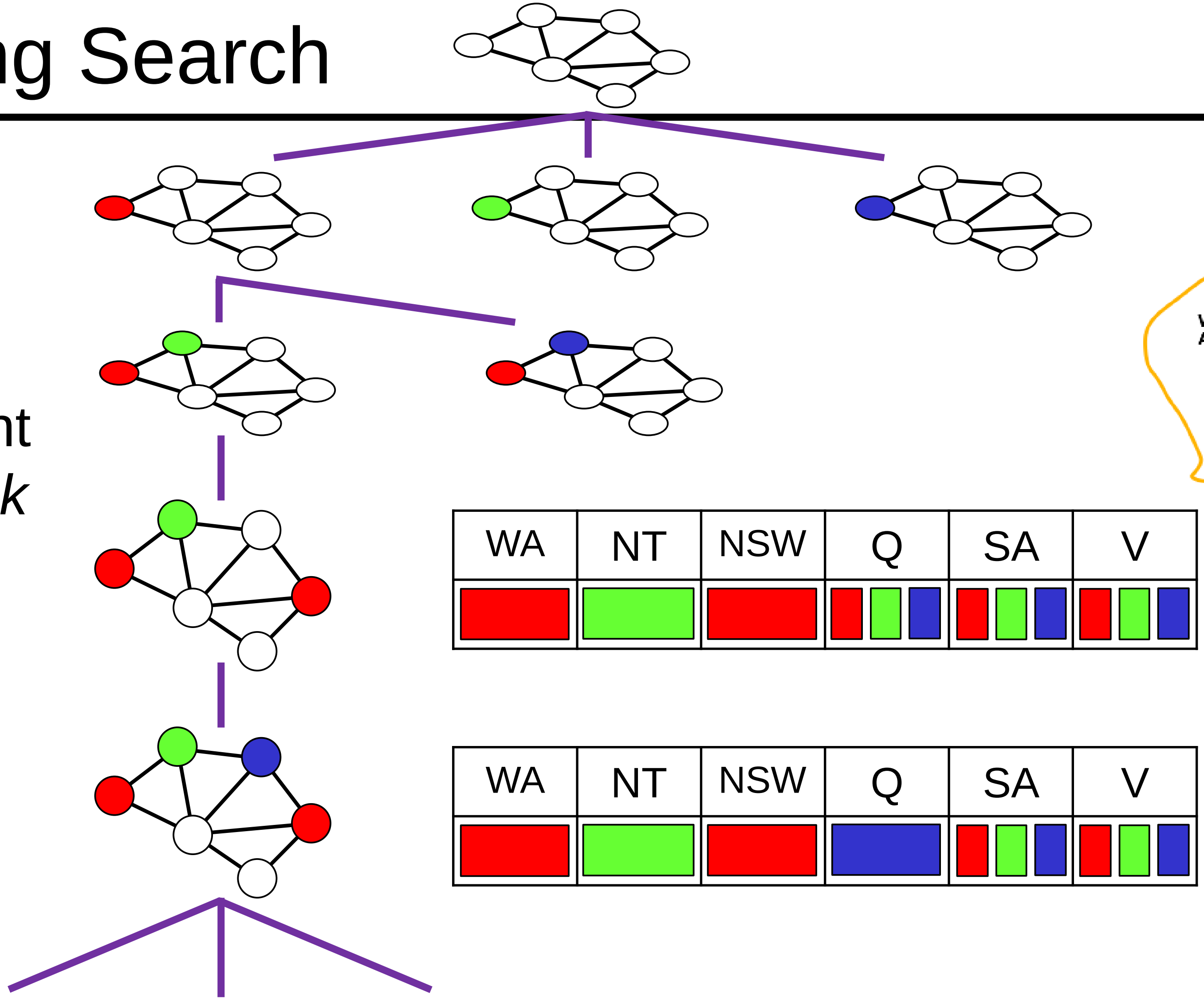


WA	NT	NSW	Q	SA	V

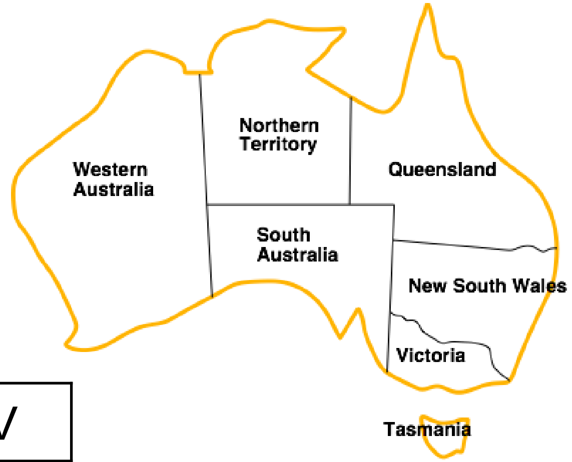
WA	NT	NSW	Q	SA	V



Backtracking Search



If no consistent assignment at current depth, then *backtrack* to previous depth



No legal assignments for SA → *Backtrack!*

4 Queens Example

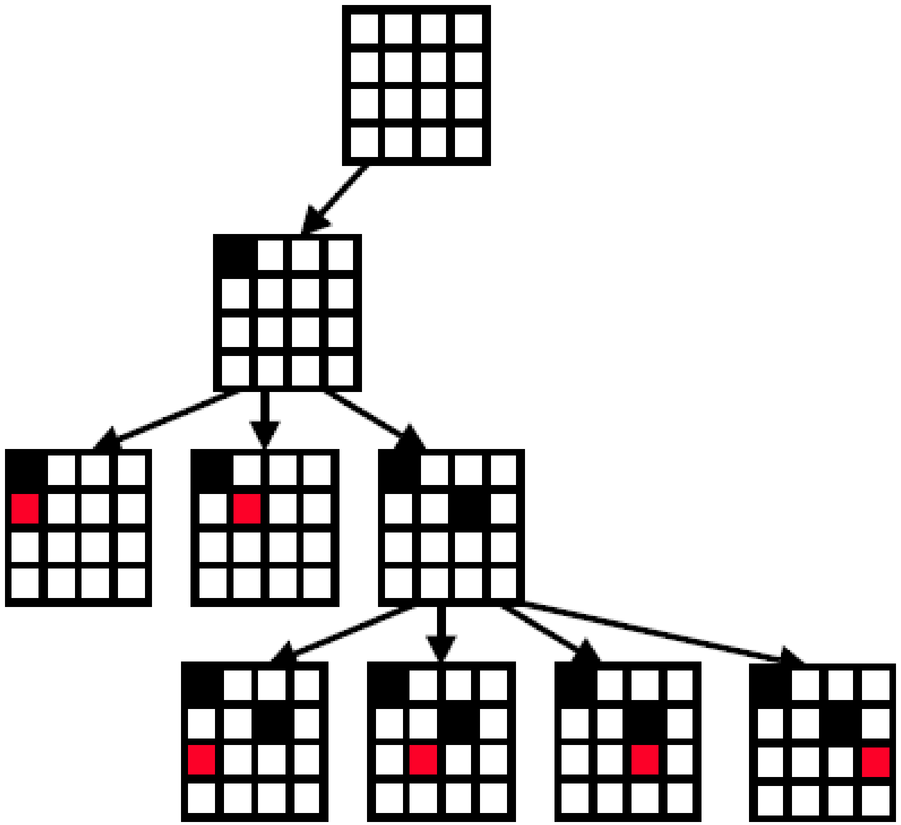


Figure credit: Hajjat Ghaderi, U. Toronto

4 Queens Example

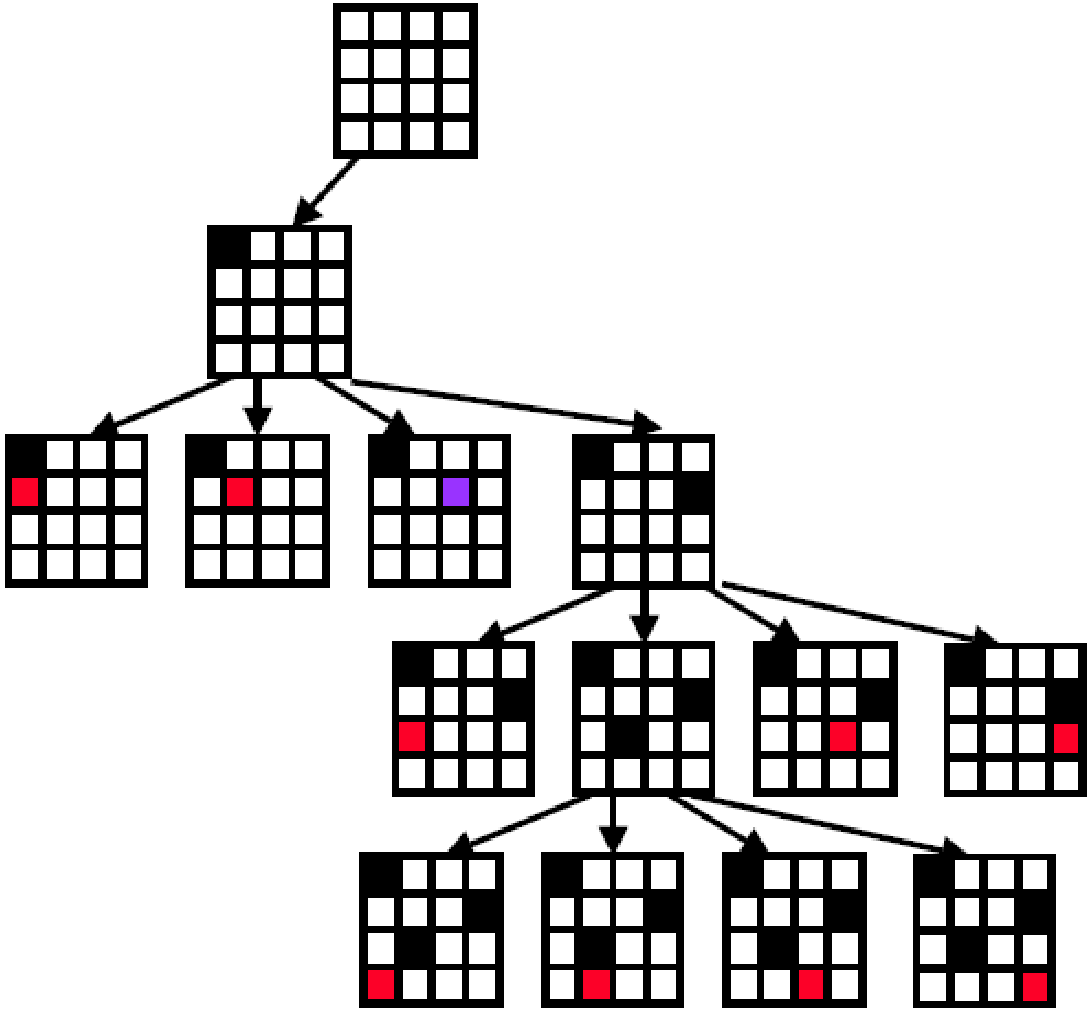


Figure credit: Hajjat Ghaderi, U. Toronto

Pseudocode for Backtracking Search

function BACKTRACK(*assignment*, *csp*)

    if *assignment* is complete, **then return** *assignment*

    var ← SELECT-UNASSIGNED-VARIABLE(*csp*)

    foreach *val* **in** ORDER-DOMAIN-VALUES(*var*, *assignment*, *csp*) **do**

        if *val* is consistent with *assignment* **then**

            add {*var* = *val*} to *assignment*

            inferences ← INFERENCE(*csp*, *var*, *val*)

            if inferences ≠ failure **then**

                add *inferences* to *assignment*

                result ← BACKTRACK(*assignment*, *csp*)

                if *result* ≠ *failure* **then return** *result*

        remove {*var* = *val*} and *inferences* from *assignment*

    return *failure*

Pseudocode for Backtracking Search

function BACKTRACK(*assignment*, *csp*)
  if *assignment* is complete, **then return** *assignment*
  var ← SELECT-UNASSIGNED-VARIABLE(*csp*)
  foreach *val* **in** ORDER-DOMAIN-VALUES(*var*, *assignment*, *csp*) **do**
    if *val* is consistent with *assignment* **then**
      add {*var* = *val*} to *assignment*
      inferences ← INFERENCE(*csp*, *var*, *val*)
      if inferences ≠ failure **then**
        add *inferences* to *assignment*
        result ← BACKTRACK(*assignment*, *csp*)
        if *result* ≠ *failure* **then return** *result*
    remove {*var* = *val*} and *inferences* from *assignment*
  return *failure*

Given a valid assignment, add to solution and call BACKTRACK recursively to go deeper

If you can't go deeper, remove last value and pop to previous level of tree to try again

Pseudocode for Backtracking Search

function BACKTRACK(*assignment*, *csp*)

  if *assignment* is complete, **then return** *assignment*

  var ← SELECT-UNASSIGNED-VARIABLE(*csp*)

  foreach *val* **in** ORDER-DOMAIN-VALUES(*var*, *assignment*, *csp*) **do**

    if *val* is consistent with *assignment* **then**

      add {*var* = *val*} to *assignment*

      inferences ← INFERENCE(*csp*, *var*, *val*)

      if inferences ≠ failure **then**

        add *inferences* to *assignment*

        result ← BACKTRACK(*assignment*, *csp*)

        if *result* ≠ *failure* **then return** *result*

    remove {*var* = *val*} and *inferences* from *assignment*

  return *failure*

Use heuristics to select variables and values to expand first

Use inference to check outcome of assignments, reducing search space

Ways to Improve Backtracking Search

What variable to assign next?

What order to use for values of a variable?

What inferences can be made from an assignment?

How can we take advantage of problem structure?

Ways to Improve Backtracking Search

What variable to assign next?

Choose variable with fewest values left → fail first

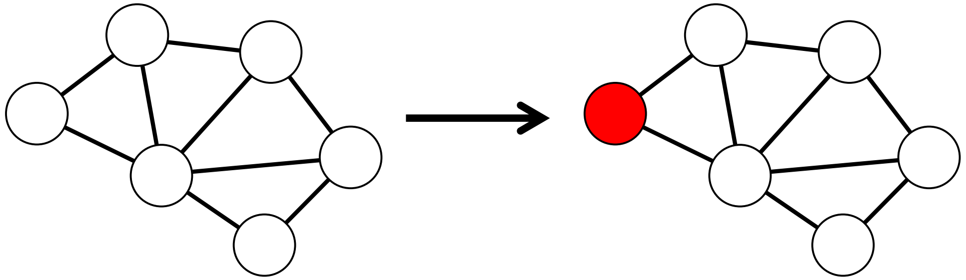
What order to use for values of a variable?

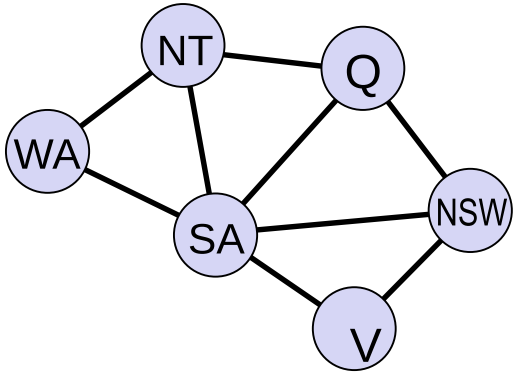
What inferences can be made from an assignment?

How can we take advantage of problem structure?

Minimum Remaining Values (MRV)

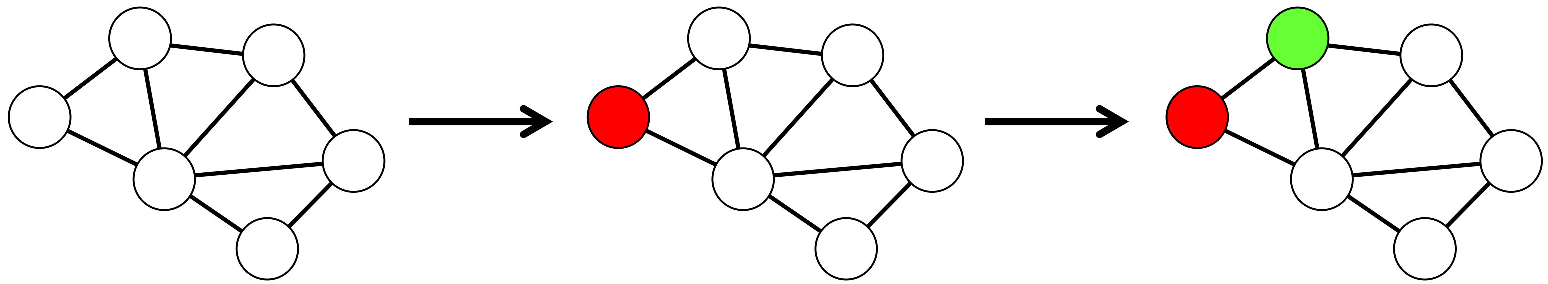
Choose the variable with the fewest remaining legal values

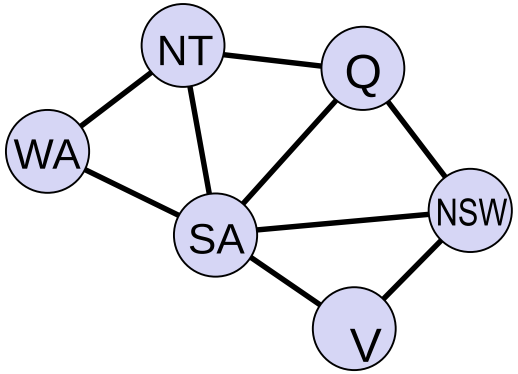




Minimum Remaining Values (MRV)

Choose the variable with the fewest remaining legal values

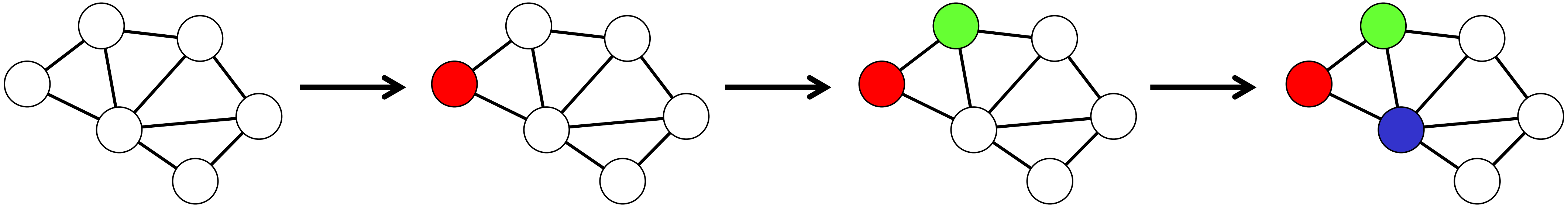


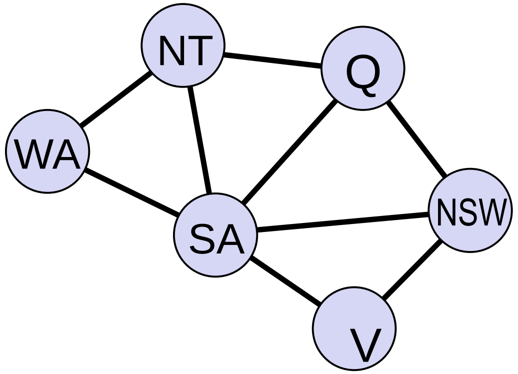


NT and SA both valid choices
Both constrained by WA = red

Minimum Remaining Values (MRV)

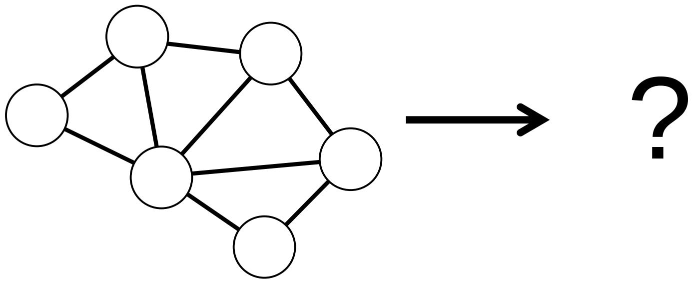
Choose the variable with the fewest remaining legal values





Minimum Remaining Values (MRV)

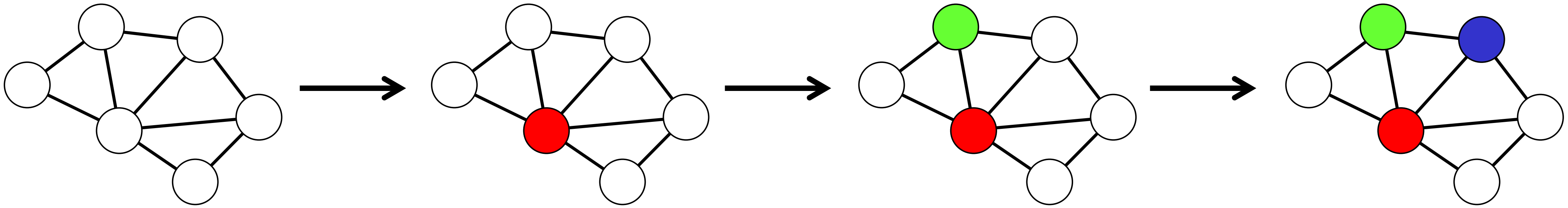
Choose the variable with the fewest remaining legal values



What should we do when all variables are equal under MRV?

Degree Heuristic: Tie Breaker for MRV

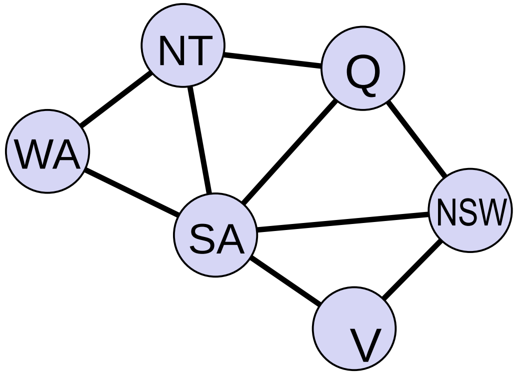
Choose the variable which participates in largest number of constraints on unassigned variables (highest degree)



SA has degree 5

NT and NSW equal with degree 2

WA and Q have 1 MRV, Q has degree 1, while WA has 0



Ways to Improve Backtracking Search

What variable to assign next?

Choose variable with fewest values left → fail first

What order to use for values of a variable?

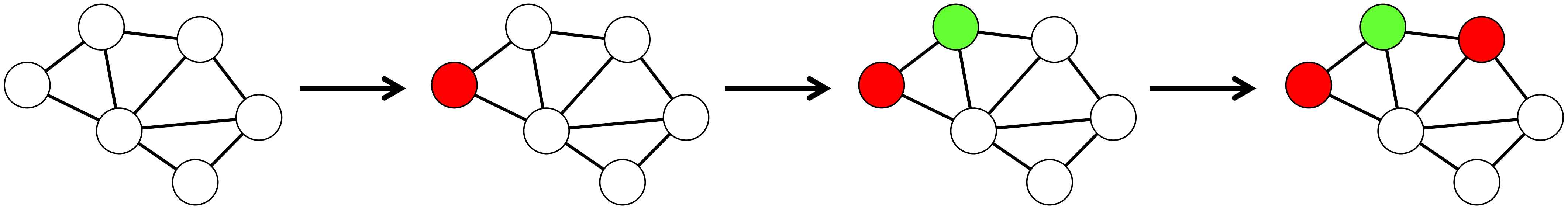
Choose val that leaves most options for remaining vars → fail last

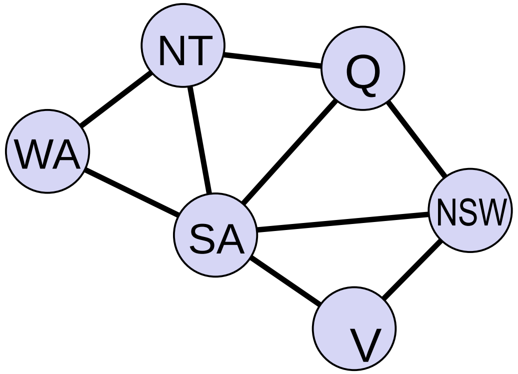
What inferences can be made from an assignment?

How can we take advantage of problem structure?

Least Constraining Value

Choose the value which removes the least number of potential values from the remaining unassigned variables





Choosing Q=red leaves 1 value for SA,
while Q=blue would leave 0 values

Summary

- Constraint Satisfaction Problems encode explicit problem structure in a simple representation language (variables + values + const)
- Backtracking search is a specialized version of DFS that exploits the structure of CSPs
- Unlike classical search, there is no path cost. The only thing that matters is reaching a consistent and complete solution quickly
- The Minimum Remaining Values heuristic for variable selection can decrease the search space, use the degree heuristic for ties
- The Least Constraining Value heuristic for value ordering can reduce the amount of backtracking

Questions?