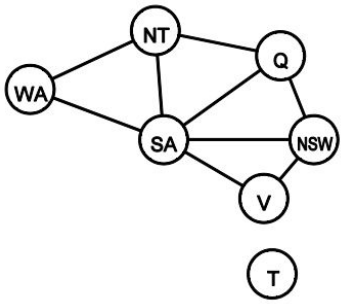
Week 6: Constraint Satisfaction Problems

- CSP examples
- Backtracking search for CSPs
- Problem structure and problem decomposition
- Local search for CSPs

Constraint Satisfaction Problem:
state is defined by *variables* X, with *values* from a *domain* D.

Constraint graph:



Discrete variables:
Takes $O(d^n)$ to complete.
d = domain size
n = number of variables
Continuous variables:
Linear constraints solvable in polynomial time.

Constraint varieties:

Unary	SA != green
Binary	SA != VIC
Higher order	3 or more variables
Preferences	(Soft constraints) red > green

Real world CSP examples:
- Who teaches what class
- Which class is offered when and where (timetabling)
- Hardware configuration
- Spreadsheets
- Transportation scheduling
- Factory scheduling
- Floorplanning

Using backtracking search instead of brute force incremental search makes our time go from $O(n!d^n)$ to $O(d^n)$

Backtracking search:

AKA Depth First Search for CSPs with single variable assignments.

Pseudocode:

```
BacktrackingSearch(csp) returns (solution or failure)
       return RecursiveBacktracking(assignment, csp);

RecursiveBacktracking(assignment, csp) returns solution/failure
       if assignment is complete then return assignment
       var <- FindUnassignedVar()
       for each value in OrderDomainValues()
              if value is consistent with Constraints(csp) then
                     add (var == value) to assignment
                     result <- RecursiveBacktracking(assignment, csp)
                     if result !- failure
                            return result
                     remove (var == value) from assignment
       return failure
```

Improving backtracking search speed:

- Which variables should be assigned next?
- In what order should its values be tried?
- Can we detect certain failures early?
- Can we take advantage of the structure of the specific problem?

Other methods:

- Degree heuristic
  - Choose the variable with the most constraints on remaining variables.
- Least constraining value
  - Choose the value that rules out the fewest values in other remaining variables.
- Forward checking
  - Keep track of remaining legal values for other remaining variables
  - Terminate when any variable runs out of legal values.
- Constraint propagation - Arc consistency
  - Arc between X & Y. Has to be a value in Y that satisfies every value in X (constraint)
  - If X loses a value, neighbors of X need to be rechecked.
  - Detects failure earlier than forward checking.
- Find independent subproblems
  - Aka tasmania.

Tree structured CSPs

Solved in $O(nd^2)$ compared to $O(d^n)$

Choose a variable as root, order from root to leaves in a line.

Nearly tree-structured CSPs

Cut a section size *c* out that makes the rest of the graph a tree. (eg remove SA)

Solved in $O(d^c (n-c) d^2)$ compared to $O(d^n)$, which is a lot faster if *c* is small.

Can also use Hill-Climbing

- Assign all variables unsatisfied constraint values.
- Reassign variable values by the min-conflicts heuristic.