CS 182 Lecture 5: Constraint Satisfaction Problems

Professors: Ariel Procaccia and **Stephanie Gil**

Email: sgil@seas.harvard.edu

Prof. Gil Office hours: Wednesdays 2:30-3:30p

Last Time

- Motion planning problems
  - Search and their relation to motion planning
  - Problems of discretization of free space
  - Questions of optimality

This Time

- Constraint Satisfaction Problems
  - Search comes up again!
  - What are CSPs useful for?
  - How to define search for this class of problems?

Planning and Identification

Planning: A sequence of actions

- The path to the goal is important
- Paths have various costs and depths
- Heuristics give problem-specific guidance

Identification: Assignments to variables

- The goal itself is important, not the path
- CSPs are specialized for identification problems
- This is still a search problem

Constraint Satisfaction Problems (CSP)

Defined by a set of

- **Variables** $X_1, X_2, \ldots, X_n$
- **Constraints** $C_1, C_2, \ldots, C_m$
- **Assignment** $\{X_i = v_i, X_j = v_j, \ldots\}$

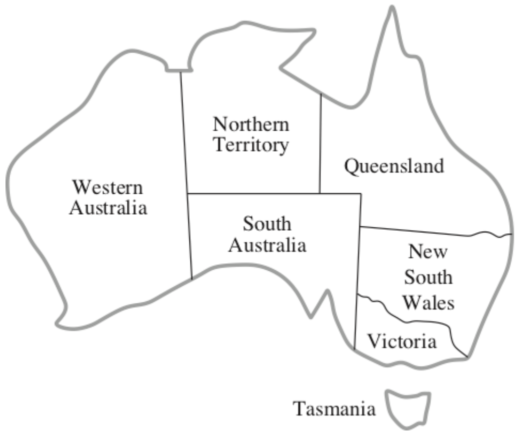
- State – defined by variables X_i with values from a domain D

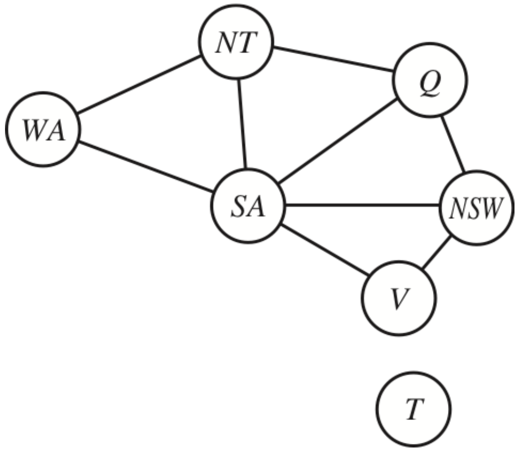
- Goal test – Set of constraints specifying allowable combinations of values for subsets of variables

A **Solution** to a CSP is a complete assignment that satisfies all constraints

Representation of a CSP

- Before we saw how representation of a problem as a graph helped us to formulate search over routes
- Now we will use graphs to help us formulate constraints



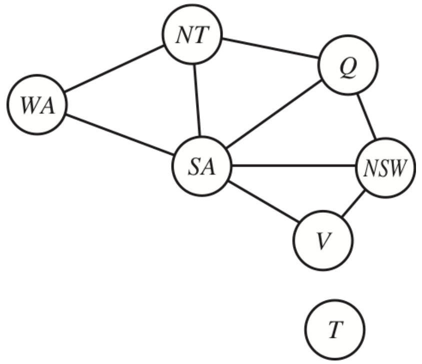


- Pairwise constraints between variables are denoted as an edge

CSP: Comparison to What We’ve Seen

- **Initial state:** the empty assignment { }, in which all variables are unassigned
- **Successor function:** a value can be assigned to any unassigned variable, provided that it does not conflict with previously assigned variables
- **Goal test:** the current assignment is complete
- **Path cost:** a constant cost (e.g. 1) for every step

Example: Graph Coloring



Ways to describe constraints:

- Variables: WA, NT, Q, NSW, V, SA, T
- Domains: D={red, green, blue}
- Constraints: Adjacent regions must have different colors
  - Implicit: WA $\neq$ NT
  - Explicit: (WA, NT) $\in$ {(red, green), (red, blue),…}

Types of Constraints

- **Continuous domain** – Examples: scheduling of experiments on the Hubble Space Telescope
  - Linear programming
  - Quadratic programming…
- **Unary constraint** – Restricts the value of a single variable. Example: $< (SA), SA \neq \ green >$
- **Binary constraint** – Relates two variables. Example: $SA \neq \ NSW$
- **Global constraint** – A constraint involving all the variables in a problem. Example: *Alldiff* which says that all of the variables involved in the constraint must have different values

Example: Cryptarithmetic Problems

- <u>Idea: </u>assign every letter a unique digit. The result should satisfy the predefined arithmetic rules

```
   SEND        Variables
+  MORE        needing
               assignment
-------------
  MONEY        Arithmetic
               constraint
```

$S \rightarrow 9$
$M \rightarrow 1$

```
   S
+  M
-------------   Produces a new
  MO            assignment O=0 also
```

➢ This does not capture the whole solution however, because all constraints should be satisfied…

Example with Explicit Constraints

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

Try:
R=2
O=6
C_3=1

- <u>Variables:</u> F,T,U,W,R,O,C_1,C_2,C_3
- <u>Domain:</u> {0,1,2,3,4,5,6,7,8,9}
- <u>Constraints:</u> *Alldiff* (F,T,U,W,R, O,C_1,C_2,C_3)

$$O + O = R + 10C_{10}$$
$$C_{10} + W + W = U + 10C_{100}$$
$$C_{100} + T + T = O + 10C_{1000}$$
$$C_{1000} = F$$

Types of Constraints

- **Continuous domain** – Examples: scheduling of experiments on the Hubble Space Telescope
- **Unary constraint** – Restricts the value of a single variable. Example: $< (SA), SA \neq \ green >$
- **Binary constraint** – Relates two variables. Example: $SA \neq \ NSW$
- **Global constraint** – A constraint involving all the variables in a problem. Example: *Alldiff*

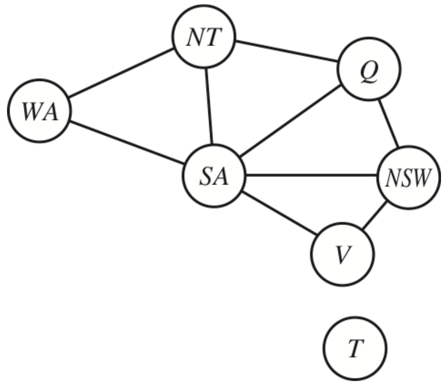
➢ Every finite-domain constraint can be reduced to a set of binary constraints if enough auxiliary variables are introduced – *so we could transform any CSP into one with only binary constraints!* This makes the algorithms simpler

Example: Graph Coloring

Ways to describe constraints:

- Constraint (English description): Adjacent regions must have different colors
- Implicit constraints: WA $\neq$NT
- Explicit constraints: (WA,NT) $\in$\{ (red,green),(red,blue),…\}

Example: Graph Coloring



Solution: Assignments satisfying all constraints {WA= red, NT=green, Q=red, NSW=green, V=red, SA=blue, T=green}

Example of an invalid solution?

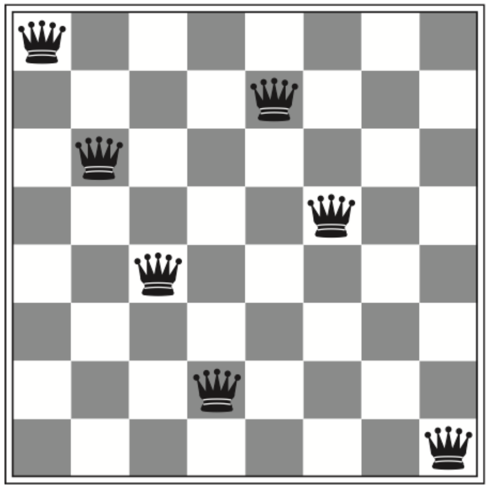
{WA= red, NT=red, Q=red, NSW=green, V=red, SA=blue, T=green}

What about

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

Another Example: N Queens

Is this a valid solution?



- Variables: X_{ij} The state of each square
- Domains: {0,1} What is happening on the square? Queen or no queen?
- Constraints: How to encode that queens cannot be on the same row or diagonal?
  - Explicit:

Constraints on queen placement

$$\forall i,j,k \quad (X_{ij}, X_{ik}) \in \{(0,0),(0,1),(1,0)\}$$ No two queens in same row

$$\forall i,j,k \quad (X_{ij}, X_{kj}) \in \{(0,0),(0,1),(1,0)\}$$ No two queens in same col

$$\forall i,j,k \; (X_{ij}, X_{i+k,j+k}) \in \{(0,0),(0,1),(1,0)\}$$
$$\forall i,j,k \; (X_{ij}, X_{i+k,j-k}) \in \{(0,0),(0,1),(1,0)\}$$ No two queens in same diag

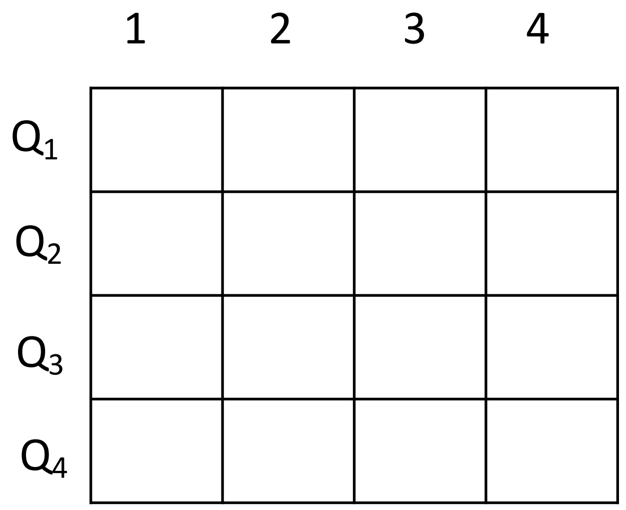
  - Implicit constraint:

$$\sum_{ij} X_{ij} = N$$

There must be N queens on the board!

➢ This constraint prevents the trivial solution of no queens on the board

N Queens: Different Formulation



- Variables: Q_k

- Domains: {1,2,3,…,N}

- Constraints
  - Explicit: $(Q_{1,}Q_2) \in \{(1{,}3),(1{,}4),\ldots\}$

- Draw a constraint graph for N queens

Types of CSPs and Solution Time

- Finite domains (the examples we’ve seen so far)
  - Size d means O(d^n) complete assignments
- Infinite domains (integers, strings, etc)
  - Job scheduling where the variables are start/end times for each job
  - Linear constraints are solvable
- Continuous variables
  - Start/end times for Hubble telescope operations
  - Linear constraints are solvable in polynomial time (the subject of linear programming LP)

Some Real World CSPs

- Assignment problems: e.g. who teaches class
- Timetable problems: e.g. which class is offered when and where?
- Transportation scheduling
- Factory scheduling
- Fault diagnosis…

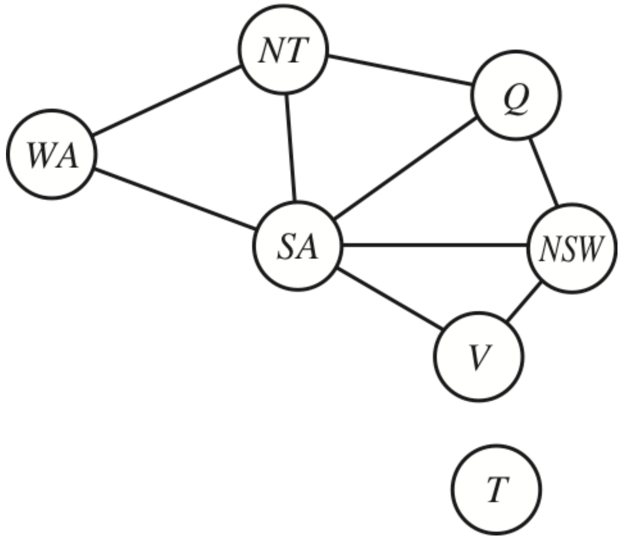
Standard Search Formulation

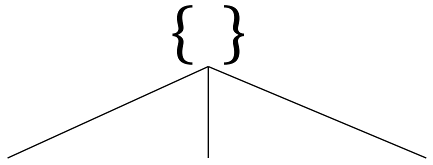
A standard template. Compare to what we've seen so far…

- States – defined by the values assigned so far
- Initial state – the empty assignment, { }
- Successor function – assign a value to an unassigned value
- Goal test – the current assignment is complete and satisfies all constraints

Example: Graph Coloring

Q1: What would a BFS assignment of variables to values look like for this problem?





Thinking About Search

- Q2 (polls everywhere): Would DFS do better?
  - Yes
  - No

Solving CSPs: What Affects Efficiency?

1) Which variable is assigned next and in what order?

2) What are the implications of the current variable assignments on other unassigned variables

3) When a path fails, can search avoid repeating this failure in subsequent paths?

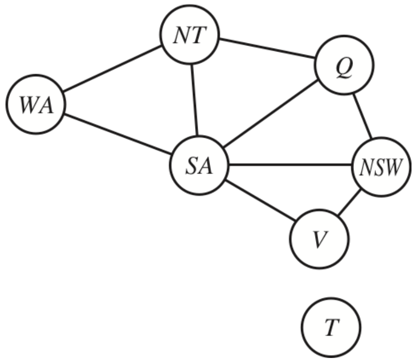
Solving CSPs: Backtracking

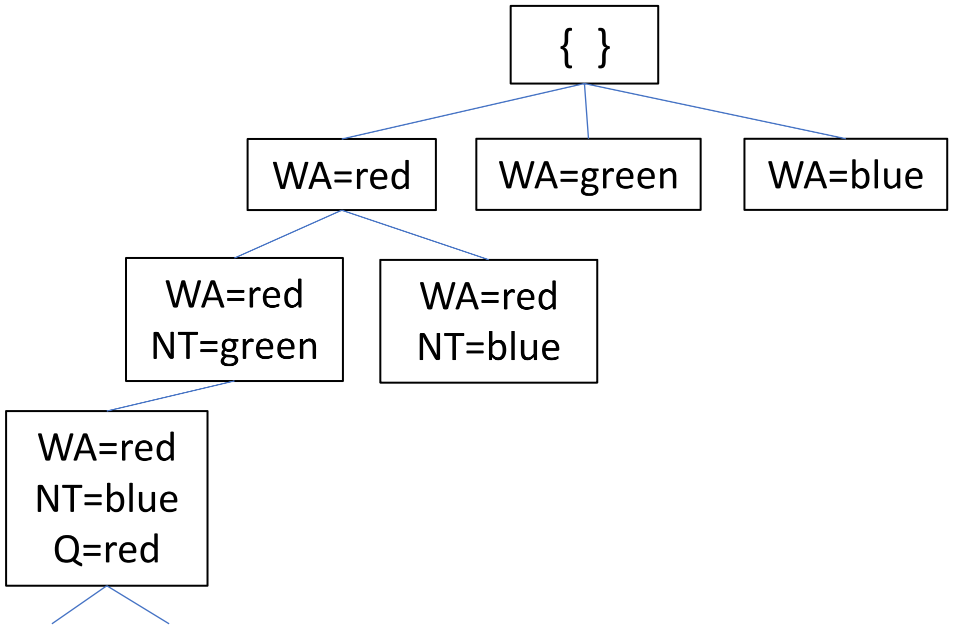
<u>Variables:</u>

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

<u>Domain</u>: {red, green, blue}

- Try BFS on the map coloring problem
- What about DFS?
  - Backtracking search is the basic uninformed algorithm for solving CSPs





Revisit Ideas for Efficiency

Keys for speeding up the search:

- Ordering: which variable should be assigned next?
    - In what order should its values be tried?

- Filtering: can we detect inevitable failure early?

- Structure: can we exploit problem structure?

Algorithms for solving CSPs that we will see next build upon these ideas for speeding up solution finding!

Forward Checking

- A type of filtering
- Idea: keep track of domains for unassigned variables and cross off bad options
  - Cross off values that violate a constraint when added to the existing assignment

Step 1: Assign the first variable. Remove conflicting values from other variable domains using the constraint graph {WA=red}

Step 2: Continue to the next variable. Again, remove conflicting values from remaining domains {WA=red, Q=green}

Step 3: Assign next variable. If a domain is left empty, backtrack. (Don’t wait to get to the illegal assignment) {WA=red, Q=green, V=blue}

Forward Checking (cont.)

- What this looks like for our Australia example

	WA	NT	Q	NSW	V	SA	T
Initial domains	🟦🟥🟩	🟦🟥🟩	🟦🟥🟩	🟦🟥🟩	🟦🟥🟩	🟦🟥🟩	🟦🟥🟩

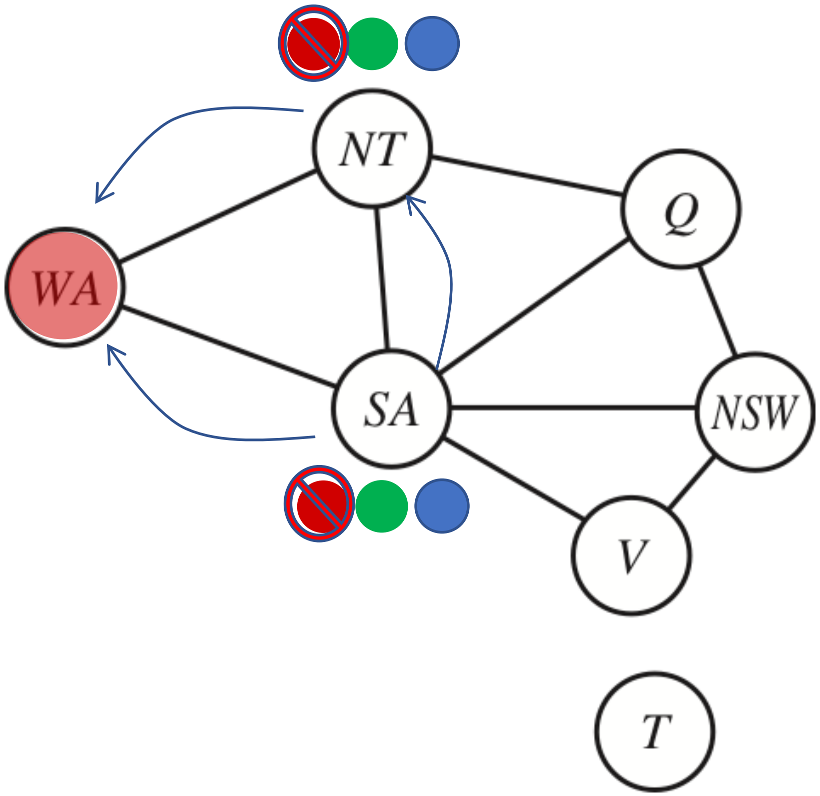
Name of the Game: Fail Fast

Borrowing some startup philosophy:

> “Fail fast is a philosophy that values extensive testing and incremental development to determine whether an idea has value. An important goal of the philosophy is to cut losses when testing reveals something isn’t working and quickly try something else”

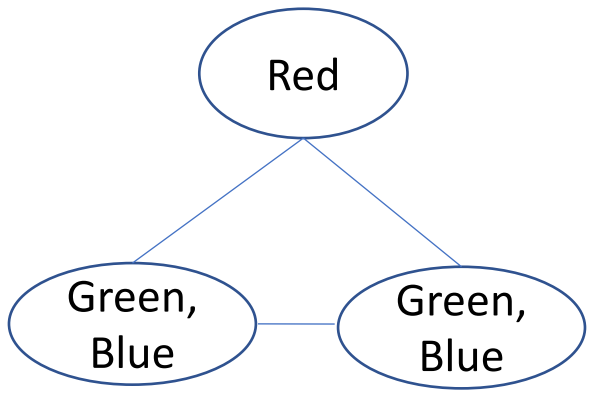
Arc Consistency

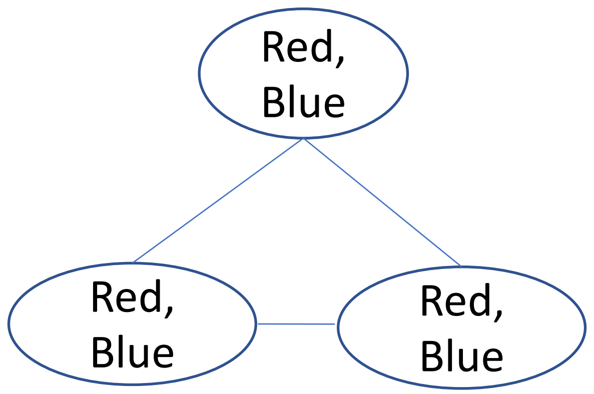
- Extend forward checking concept:
  - Follow constraints all the way through the constraint graph



Limitations of Arc Consistency

- Arc consistency does not detect all failures





Is this arc consistent?

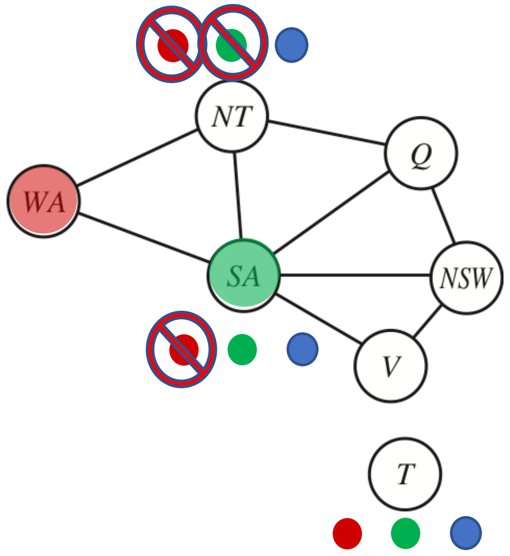
Are there solutions remaining?

Is this arc consistent?

Are there solutions remaining?

Minimum Remaining Values (MRV)

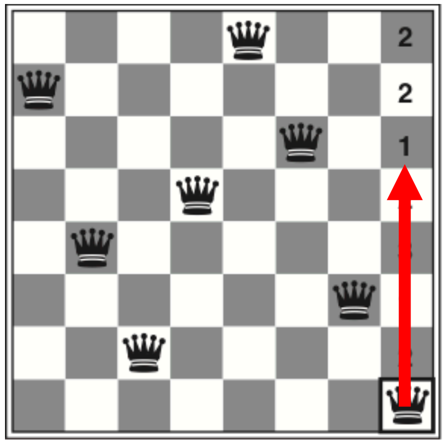
- How else can we “fail fast”?
  - Variable ordering



- Minimum Remaining Values (MRV)
  - Choose the next variable for assignment as the one with the fewest number of legal values left in its domain

Local Search

- Idea: start with an assignment



- Successor function picks one queen and considers moving it elsewhere in its column
- Heuristic for choosing a new value for the variable is to select the value that results in the minimum # of conflicts with other variables
- This is the Minimum Conflicts heuristic

When to Choose Local Search?

- This is not the “fail fast method” anymore…
- When is it good to use local search?
  - Like most local search problems, it is best when we start with a good solution
  - **Example 1:** Online re-planning
  - **Example 2:** Airline schedules changing due to weather and fixing the schedule with a minimum # of changes
  - **Example 3:** This has been used to schedule observations for the Hubble telescope, reducing the time taken to schedule a week of observations from 3 weeks to 10 minutes

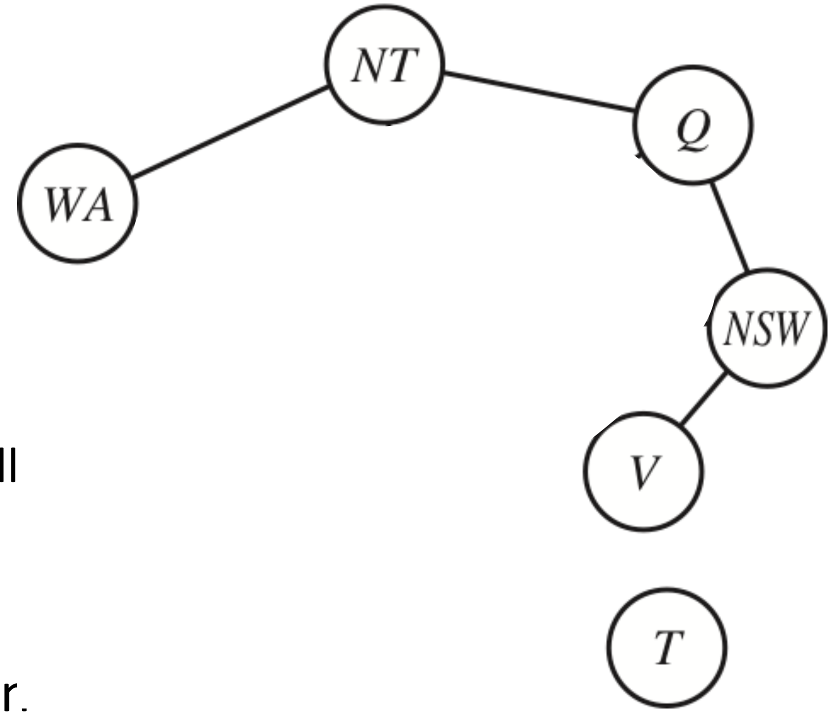
Why not use a “fail fast” approach here?

The Role of Structure

- Idea: decompose the problem into many sub-problems
- Example: Tasmania as an **independent sub-problem**
- Another example: CSP problems are connected by a single path (or the constraint graph forms a tree)
  - Any tree structured CSP can be solved in time linear in the # of variables [see Russell and Norvig text]

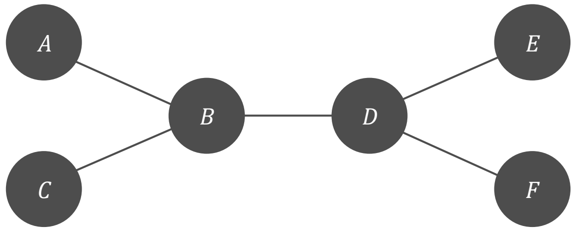
➢ We will see the concept of decompositior into sub-problems again with Dynamic Programming

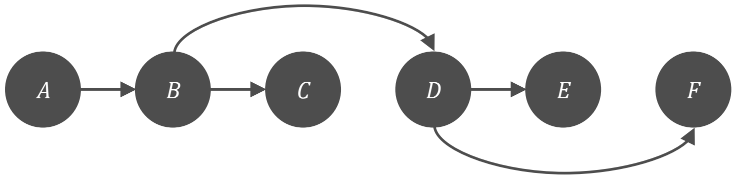
- Indeed a very common approach is to take advantage of *structure* in the problem



The Structure of Problems (cont.)

- Solving a CSP whose constraint graph is structured as a tree is easy!
- **Directionally arc-consistent** – A particular ordering where variables $X_1,\ldots,X_n$ satisfy X_i is arc-consistent with X_j for every $i<j$





TREE CSP SOLVER

```
function TREE-CSP-SOLVER($csp$)
    $n \leftarrow$ number of variables in $X$
    $assign \leftarrow \emptyset$
    $root \leftarrow$ any variable in $X$
    $X \leftarrow$ TOPOLOGICAL-SORT($X, root$)
    for $j = n$ down to 2 do
        MAKE-ARC-CONSISTENT(PARENT($X_j$), $X_j$)
        if consistency fails then return failure
    for $i = 1$ to $n$ do
        if $D_i$ has no consistent values then return failure
        $assign[X_i] \leftarrow$ any consistent value from $D_i$
    return $assign$
```

Running time $O(nd^2)$ for $|D_i| \leq d$

Next Time…

- Multi-robot systems!