Lecture 4: Constraint Satisfaction Problems

CS 486/686: Introduction to Artificial Intelligence

Outline

- What are Constraint Satisfaction Problems (CSPs)
- Standard Search and CSPs
- Leveraging problem structure - Improvements

Introduction

Standard search

State is a "black box": arbitrary data structure

Goal test: any function over states

Successor function: anything that lets you move from one state to another

Constraint satisfaction problems (CSPs)

A special subset of search problems

States are defined by ***variables*** X_i with values from ***domains*** D_i

Goal test is a ***set of constraints*** specifying allowable combinations of values for subsets of variables

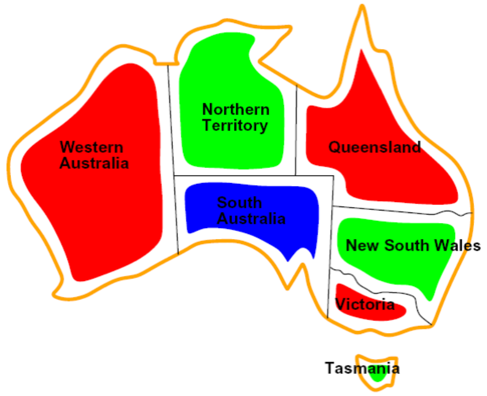
Example: Map Colouring

Variables

V={T, V, NSW, Q, NT, WA, SA}

Domains

D={red, blue, green}



Constraints: adjacent regions must have different colours

Implicit: WA≠NT

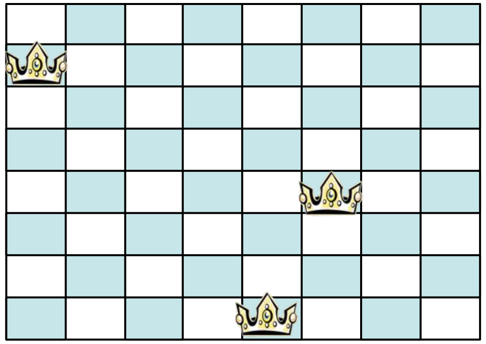
Explicit: (WA, NT)∈ {(red, blue), (red, green), (blue, red)…}

Solution is an assignment satisfying all constraints

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

N Queens Problem

Variables: $X_{i,j}$



Domains: {0,1}

Constraints:

$$\forall i, j, k \;\; (X_{ij}, X_{ik}) \in \{(0,0), (0,1), (1,0)\}$$
$$\forall i, j, k \;\; (X_{ij}, X_{kj}) \in \{(0,0), (0,1), (1,0)\}$$
$$\forall i, j, k \;\; (X_{ij}, X_{i+k,j+k}) \in \{(0,0), (0,1), (1,0)\}$$
$$\forall i, j, k \;\; (X_{ij}, X_{i+k,j-k}) \in \{(0,0), (0,1), (1,0)\}$$

N Queens Problem

Variables: Q_i

Domains: {1,2,…,N}

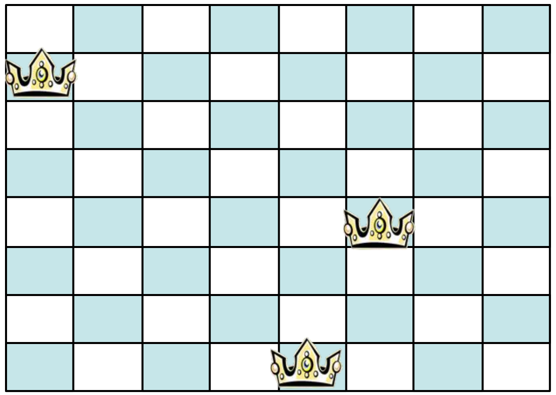
Constraints:

Implicit:

$$\forall i, j \text{ non-threatening}(Q_i, Q_j)$$

Explict:

$$(Q_1, Q_2) \quad \in \quad \{(1,3), (1,4), \ldots\}$$
$$\ldots$$



3 Sat

Variables: $V_1, \ldots, V_n$

Domains: {0,1}

Constraints:

K constraints of the form $V_i^* \vee V_j^* \vee V_k^*$ V_i^* where V_i^* is either V_i or $\neg V_i$

$$A \neg B \vee \neg C$$
$$\neg A \vee B \vee D$$
$$D \vee B \vee E$$
$$\neg A \vee \neg B \vee C$$

The canonical NP-complete problem

Types of CSPs

Discrete Variables

Finite domains

If domain has size d, then there are $O(d^n)$ complete assignments

Boolean CSPs (including 3-SAT)

Infinite domains (e.g. integers)

Constraint languages

Linear constraints are solvable but non-linear are undecidable

Continuous Variables

Linear programming (linear constraints solvable in polynomial time)

Types of CSPs

Varieties of Constraints

- **Unary constraints** involve a single variable
  - NSW≠red
- **Binary constraints** involve a pair of variables
  - NSW≠Q
- **Higher-order constraints**: involve more than two variables
  - AllDiff(V_1,…,V_n)

Soft Constraints (preferences)

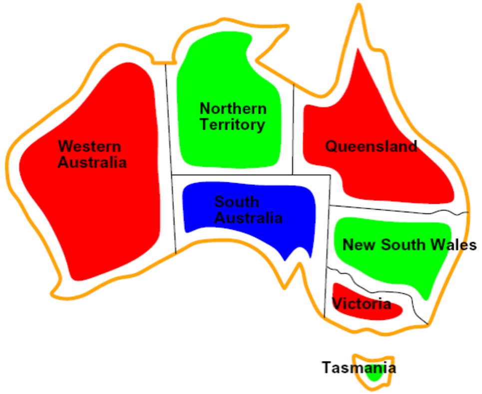
- red "is better than" green
- Constrained optimization problems

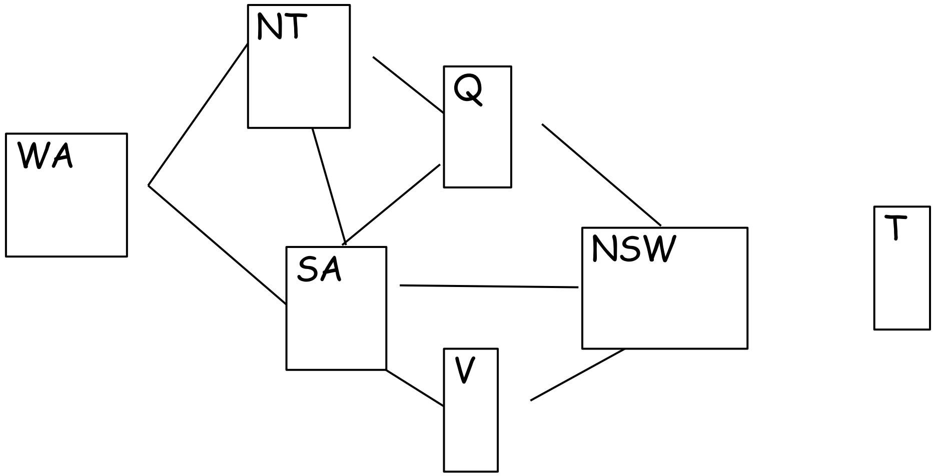
Constraint Graphs

You can represent binary constraints with a constraint graph

Nodes are variables

Edges are constraints





CSPs and Search

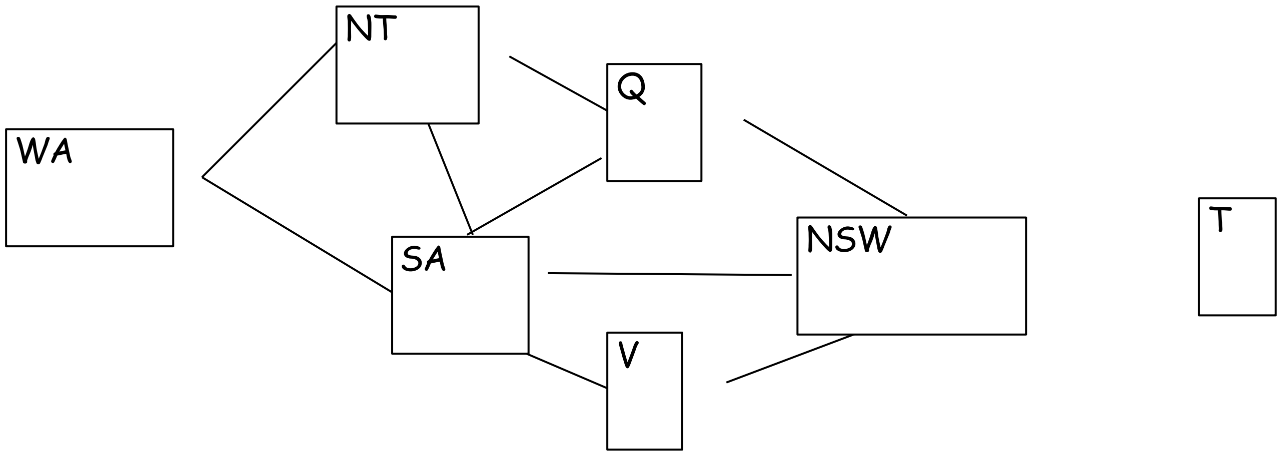
States:

Initial State:

Successor Function:

Goal Test:

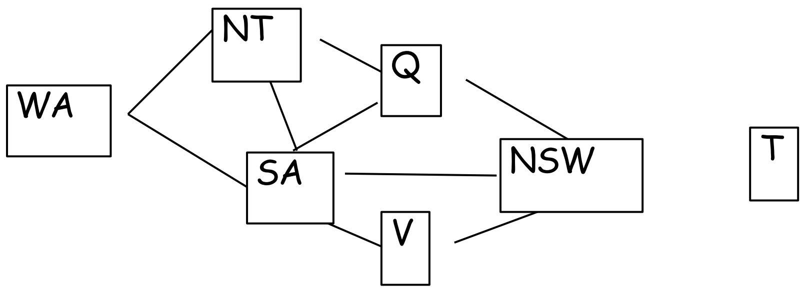
What happens if we run something like BFS?



Commutativity

Key Insight: CSPS are commutative

- Order of actions does not effect outcome
- Can assign variables in any order



{WA=red, NT=blue}
is equivalent to
{NT=blue, WA=red}

CSP algorithms take advantage of this

- Consider assignment of a single variable at each node in the tree

Backtracking Search

Backtracking search is the basic algorithm for CSPs

Select unassigned variable X

For each value $\{x_1,...,x_n\}$ in domain of X

If value satisfies constraints, assign $X=x_i$ and exit loop

If an assignment is found

Move to next variable

If no assignment found

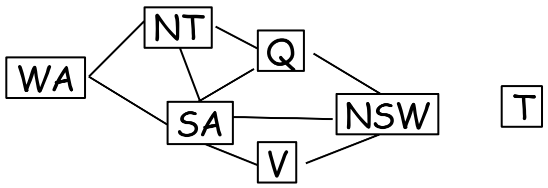
Back up to preceding variable and try a different assignment for it

One variable at a time

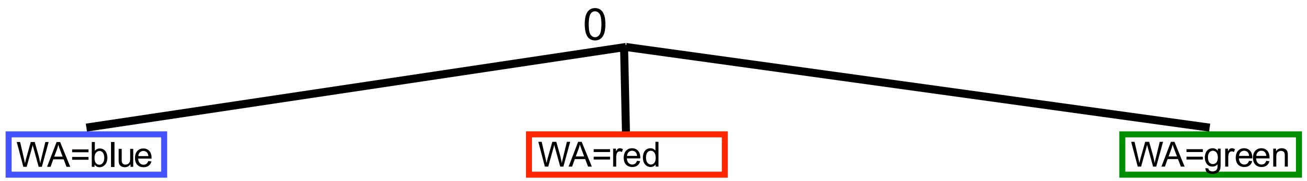
Check constraints as you go

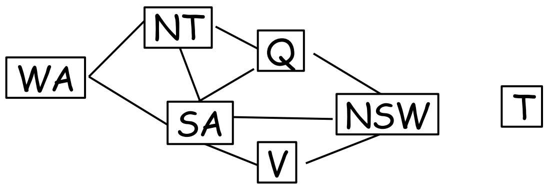
Backtracking Example

0

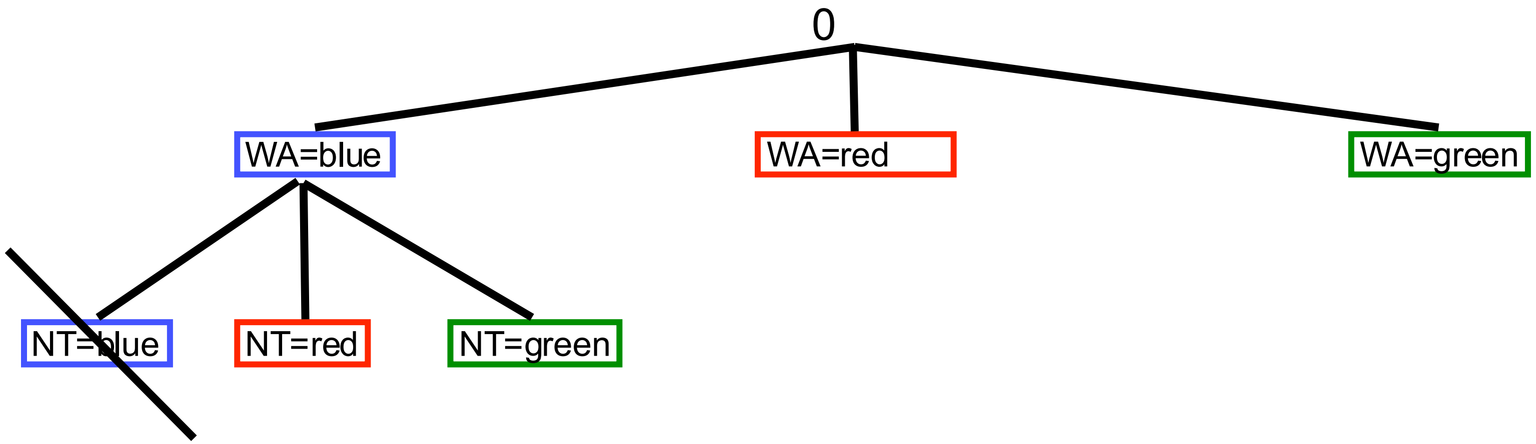


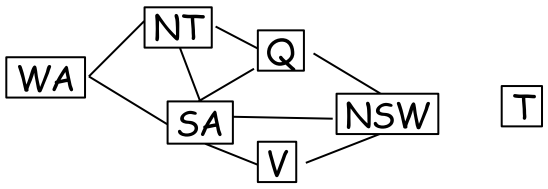
Backtracking Example



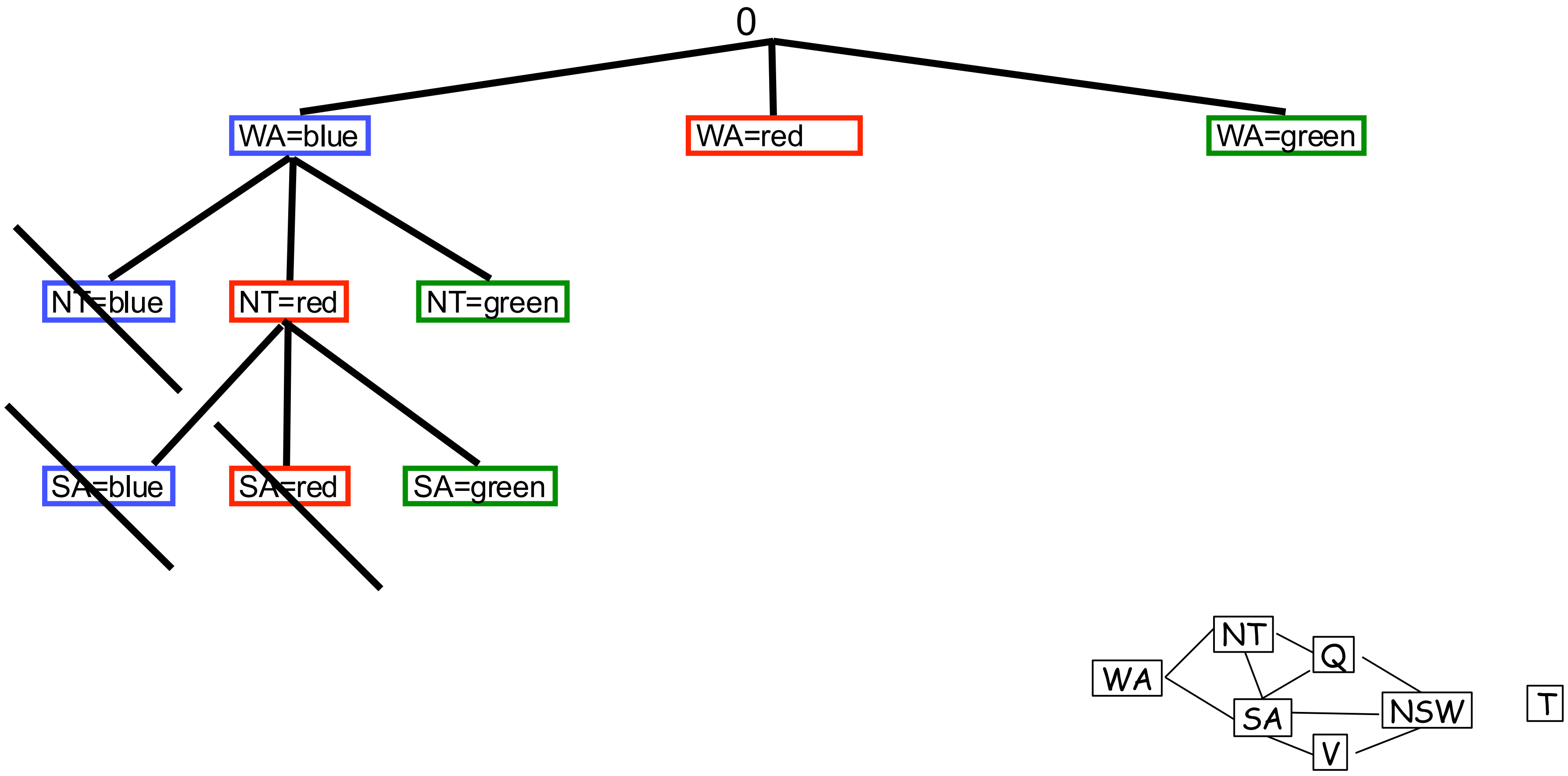


Backtracking Example





Backtracking Example



Backtracking and Efficiency

Note that backtracking search is basically DFS with some small improvements. Can we improve on it further?

Ordering:

- Which variables should be tried first?
- In what order should a variable’s values be tried?

Filtering:

- Can we detect failure early?

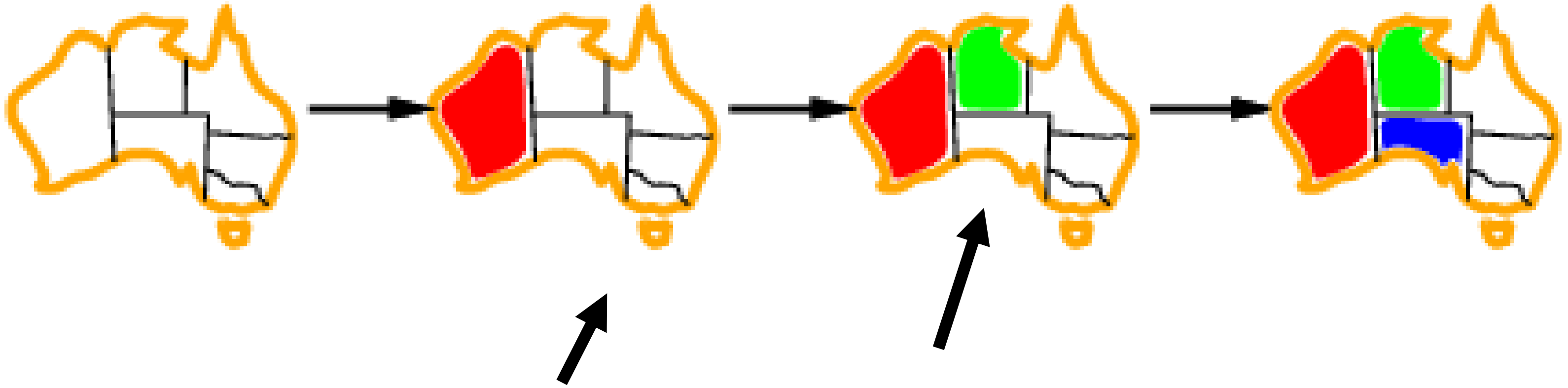
Structure:

- Can we exploit the problem structure?

Ordering: Most Constrained Variable

Choose the variable which has the fewest “legal” moves

AKA **minimum remaining values** (MRV)



D_{NT}={green, blue}

D_{SA}={green, blue}

D_{others}={red, green, blue}

D_{SA}={blue}

D_{Q}={blue, red}

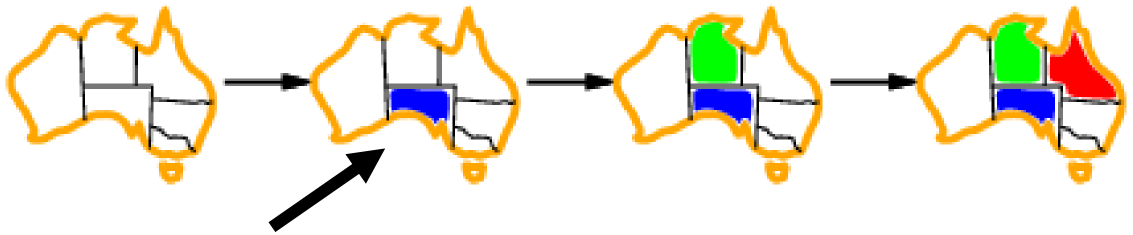
D_{others}={red,green,blue}

Ordering: Most Constraining Variable

Most constraining variable:

- Choose variable with most constraints on remaining variables

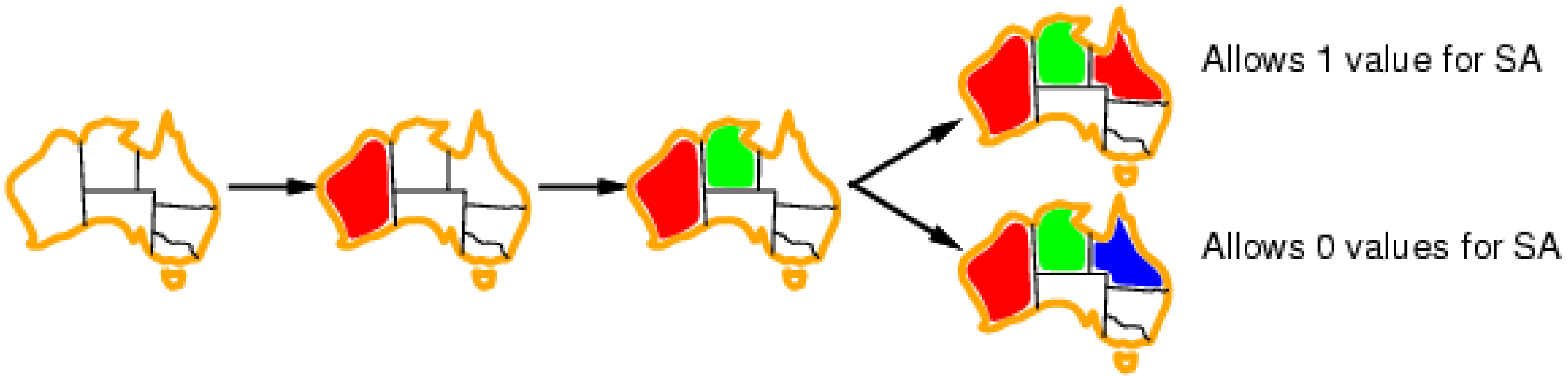
Tie-breaker among most constrained variables



Ordering: Least-Constraining Value

Given a variable, choose the least constraining value:

The one that rules out the fewest values in the remaining variables

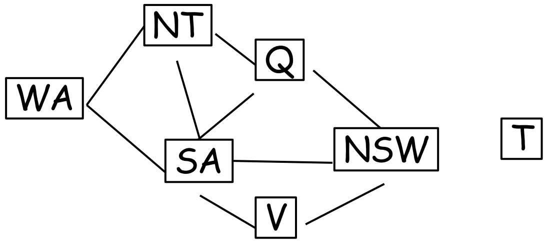


Filtering: Forward Checking

Forward checking:

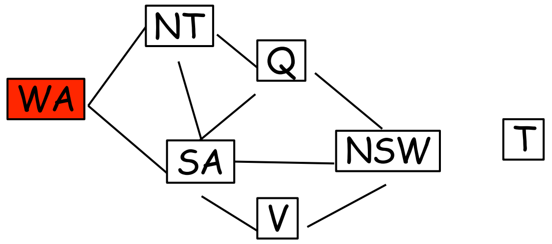
- Keep track of remaining legal values for unassigned variables
- Terminate search when any variable has no legal values

Example: Forward Checking



WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB

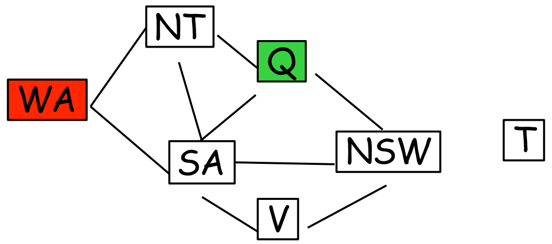
Example: Forward Checking



WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	~~R~~GB	RGB	RGB	RGB	~~R~~GB	RGB

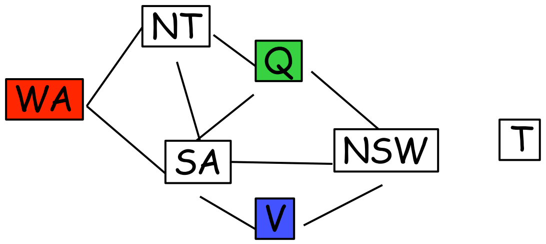
Forward checking removes the value Red of NT and of SA

Example: Forward Checking



WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	GB	RGB	RGB	RGB	GB	RGB
R	~~G~~B	G	~~R~~GB	RGB	~~G~~B	RGB

Example: Forward Checking



WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	GB	RGB	RGB	RGB	GB	RGB
R	B	G	RB	RGB	B	RGB
R	B	G	~~RB~~	B	~~B~~	RGB

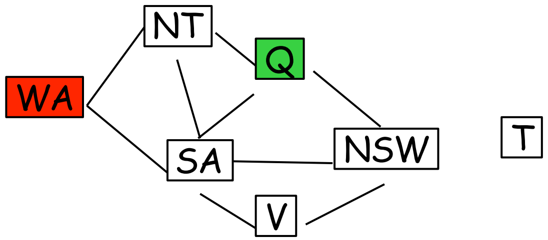
Example: Forward Checking

Empty set: the current assignment
{(WA ← R), (Q ← G), (V ← B)}
does not lead to a solution

WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	GB	RGB	RGB	RGB	GB	RGB
R	B	G	RB	RGB	B	RGB
R	B	G	R~~B~~	B	~~B~~	RGB

Filtering: Arc Consistency

Forward checking propagates information from assigned to unassigned variables, but it can not detect all future failures early



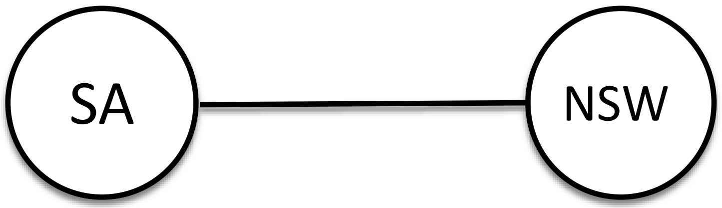
WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	GB	RGB	RGB	RGB	GB	RGB
R	B	G	RB	RGB	B	RGB

NT and SA can not both be blue!

Need to reason about constraints

Filtering: Arc Consistency

Given domains D_1 and D_2, an arc is consistent if for all x in D_1 there is a y in D_2 such that x and y are consistent.



D_{SA}={blue} D_{NSW}={blue,red}

Is the arc from SA to NSW consistent?

Is the arc from NSW to SA consistent?

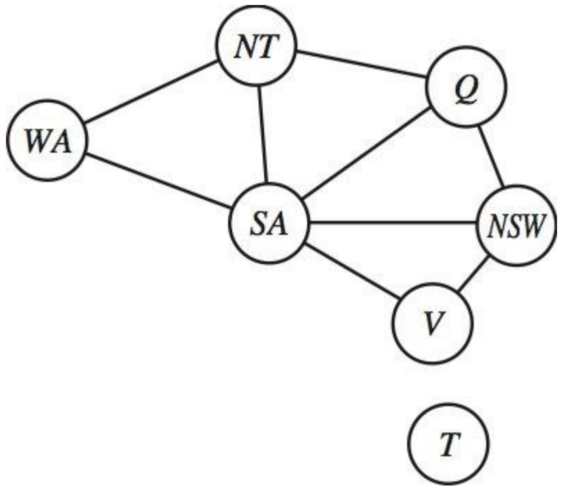
AC-3

Run as a preprocessing step to shrink variables domains

1. Initialize a queue: Put every directed arc into the queue (e.g. if X and Y share a constraint, (X,Y) and (Y,X) are added to queue)
2. Pop and check: Pop (X,Y) from queue
3. Revise Domain: For every x in Dom(X), check if there is any y in Dom(Y) that satisfies constraint. If not, delete x from Dom(x)
4. If Dom(X) is empty, exit. No solution
5. Update queue: If Dom(x) changed in Step 3, re-add all neighbouring arcs pointing back to X (e.g (Z,X))
6. Repeat until the queue is empty

If it terminates with an empty queue, then solve the CSP with the reduced domains. This is a smaller search problem than the original.

Structure: Independent Subproblems



Tasmania does not interact with the rest of the problem

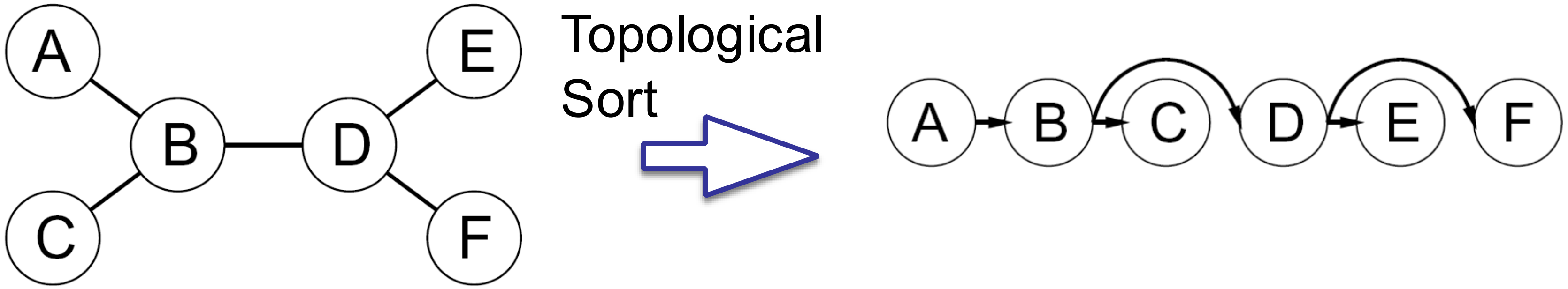
Idea Break down the graph into its connected components. Solve each component separately.

Significant potential savings

- Assume n variables with domain size d: $O(d^n)$
- Assume each component involves c variables (n/c components) for some constant c: $O(d^c\ n/c)$

Structure: Tree Structures

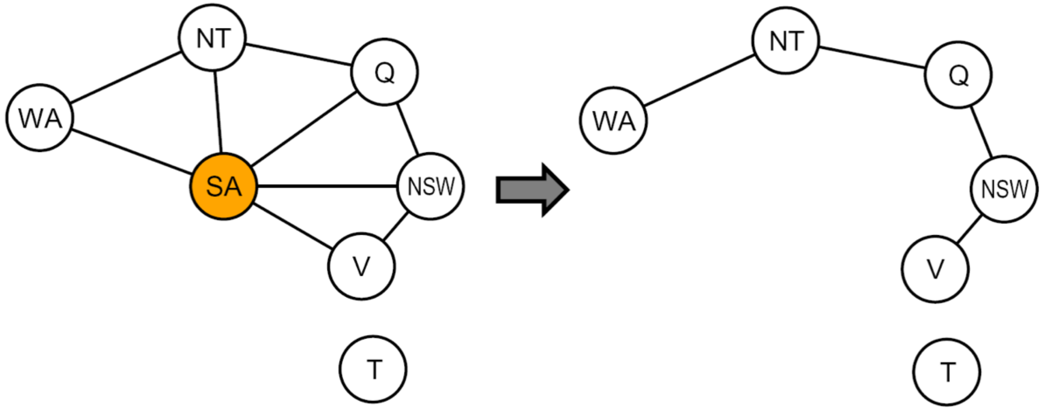
CSPs can be solved in $O(nd^2)$ if there are no loops in the constraint graph



Step 1: For i=n to 1, make-consistent(X_i,parent(X_i))

Step 2: For i=1 to n, assign value to X_i consistent with parent(X_i) [Note: No backtracking!]

Structure: Non-Trees?



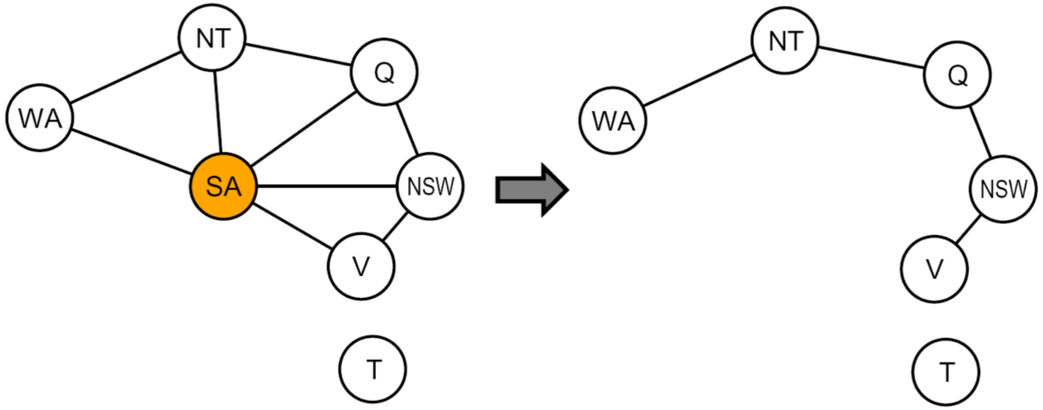
If we assign SA a colour and then remove that colour from the domains all other variables, then we have a tree

Step 1: Choose a subset S of variables such that the constraint graph becomes a tree when S is removed (S is the cycle cutset)

Step 2: For each possible valid assignment to the variables in S

1. Remove from the domains of remaining variables, all values that are inconsistent with S
2. If the remaining CSP has a solution, return it

Structure: Cutsets

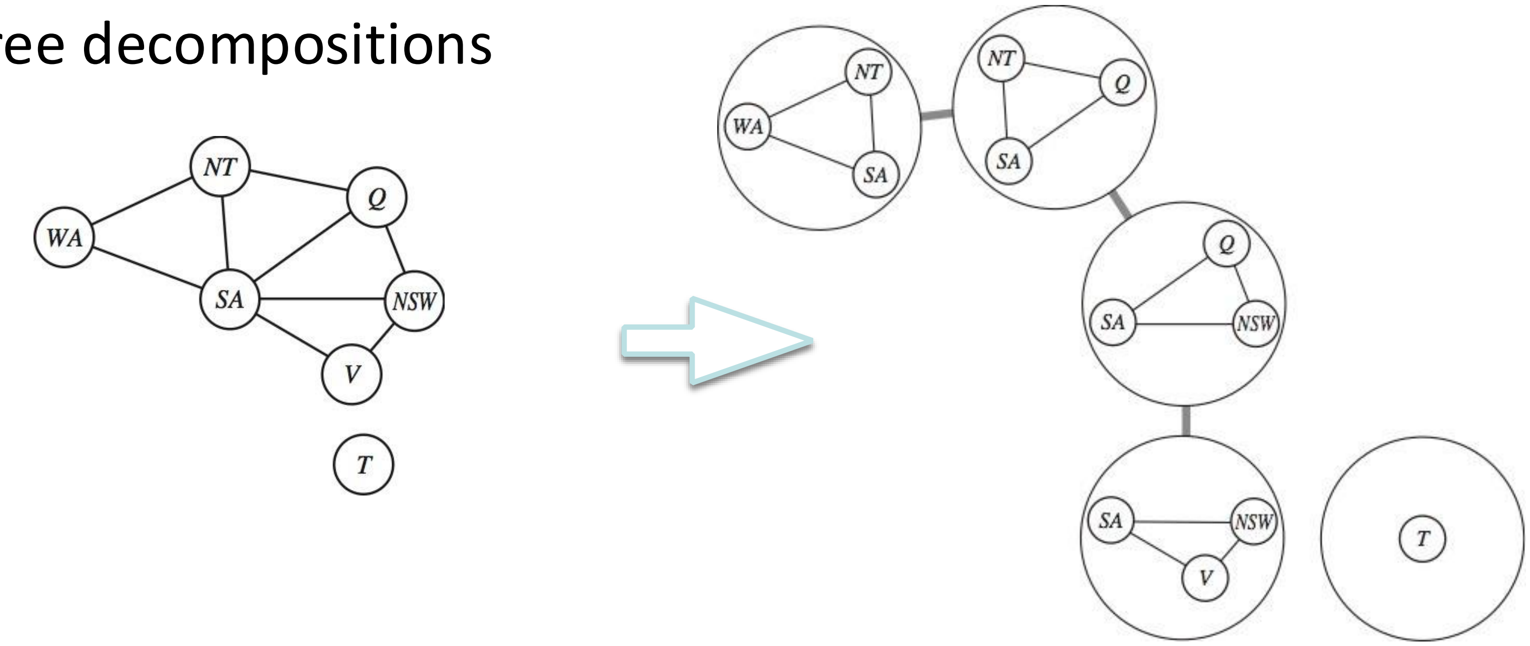


Running time:

- Let c be the size of the cutset then
  - d^c combinations of variables in S
    - For each combination must solve a tree problem of size n-c $(O(n-c)d^2)$
    - Therefore, running time is $O(d^c(n-c)d^2)$
- Finding smallest cutset is NP-hard but efficient approximations exist

Structure: Non-Trees?

Tree decompositions



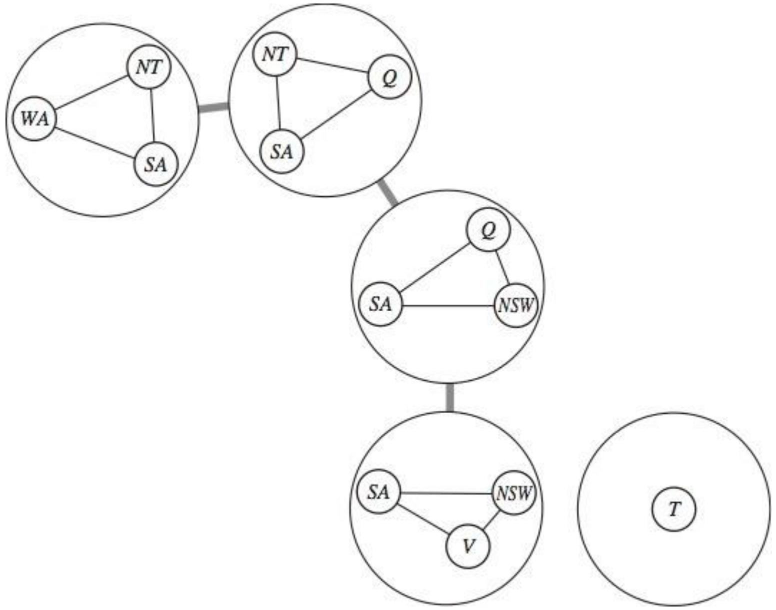
1. Each variable appears in at least one subproblem
2. If two variables are connected by a constraint, then they (and the constraint) must appear together in at least one subproblem
3. If a variable appears in two subproblems in the tree, it must appear in every subproblem along the path connecting those subproblems

Structure: Tree Decompositions

Solve each subproblem independently

e.g {(WA=r,NT=g,SA=b),(WA=b, NT=g,SA=r),…}

Solve constraints connecting the subproblems using tree-based algorithm (to make sure that subproblems with shared variables agree)



Want to make the subproblems as small as possible!
Tree width: w= Size of largest subproblem-1
Running time $O(nd^{w+1})$

Finding tree decomposition with min tree-width is NP-hard, but good heuristics exist

Summary

- Formalize problems as CSPs
- Backtracking search
- Improvements using
  - Ordering
  - Filtering
  - Structure