Artificial Intelligence

3. Constraint Satisfaction Problems

Lars Schmidt-Thieme

Information Systems and Machine Learning Lab (ISMLL)
Institute of Economics and Information Systems
& Institute of Computer Science
University of Hildesheim
http://www.ismll.uni-hildesheim.de

1. Constraint Satisfaction Problems

2. Backtracking Search

3. Local Search

4. The Structure of Problems

Problem Definition

A **constraint satisfaction problem** consists of

variables $X_1, X_2, \ldots X_n$ with **values** from given domains $\operatorname{dom} X_i$ $(i = 1, \ldots .n)$.

constraints $C_1, C_2, \ldots, C_m$ i.e., functions defined on some variables $\operatorname{var} C_j \subseteq \{X_1, \ldots, X_n\}$:

$$C_j : \prod_{X \in \operatorname{var} C_j} \operatorname{dom} X \rightarrow \{\text{true}, \text{false}\}, \quad j = 1, \ldots, m$$

Assignments

assignment: assignment A of values to some variables $\operatorname{var} A \subseteq \{X_1, \ldots, X_n\}$, i.e.,

$$A : X_3 = 7, X_5 = 1, X_6 = 2$$

An assignment A that does not violate any constraint is called **consistent** / **legal**:

$$C_j(A) = \text{true} \quad \text{for } C_j \text{ with } \operatorname{var} C_j \subseteq \operatorname{var} A, j = 1, \ldots, m$$

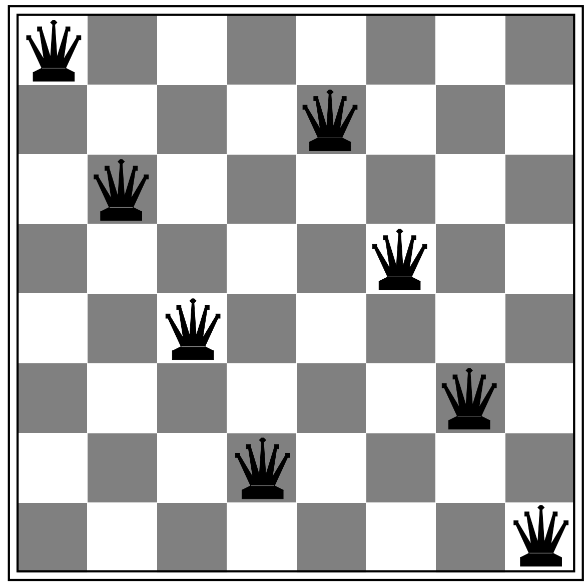
An assignment A for all variables is called **complete**:

$$\operatorname{var} A = \{X_1, \ldots, X_n\}$$

A consistent complete assignment is called **solution**.

Some CSPs additionally require an objective function to be maximal.

Example / 8-Queens



variables: $Q_1, Q_2, Q_3, Q_4, Q_5, Q_6, Q_7, Q_8$

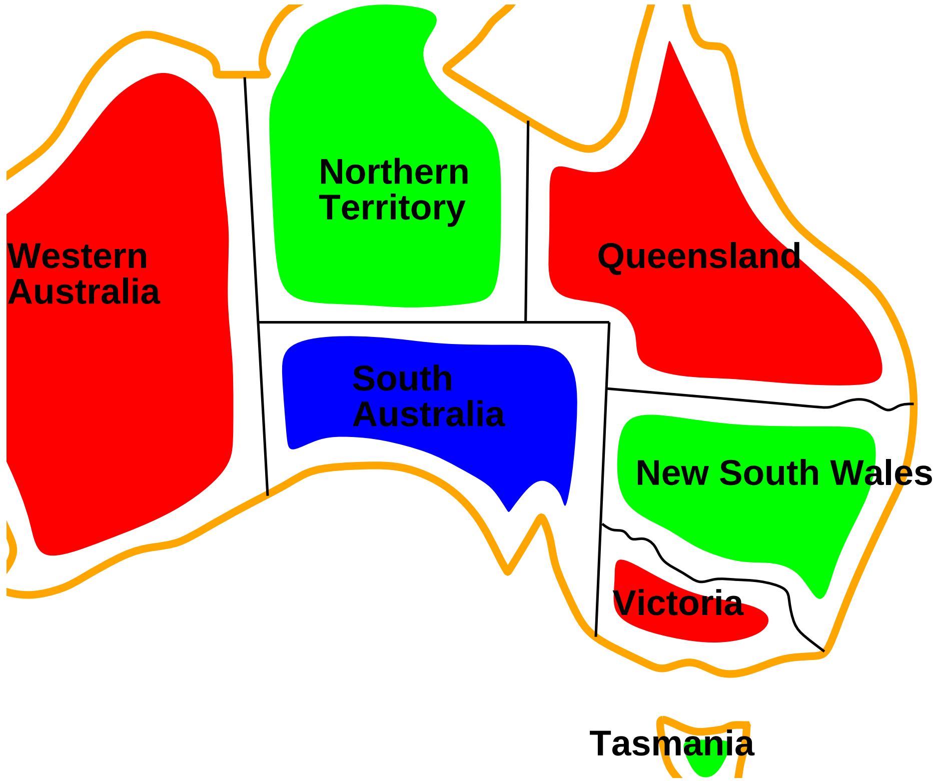
domains: $\{1, 2, 3, 4, 5, 6, 7, 8\}$.

constraints: $Q_1 \neq Q_2, Q_1 \neq Q_2 - 1, Q_1 \neq Q_2 + 1,$
$Q_1 \neq Q_3, Q_1 \neq Q_3 + 2, Q_1 \neq Q_3 - 2, \ldots$

consistent assignment:

$Q_1 = 1, Q_2 = 3, Q_3 = 5, Q_4 = 7, Q_5 = 2, Q_6 = 4, Q_7 = 6$

Example / Map Coloring



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

domains: { red, green, blue }

constraints: $WA \neq NT$, $WA \neq SA$, $NT \neq SA$, $NT \neq Q$, ...

solution:
WA = red, NT = green, SA = blue, Q = red, NSW = green, V = red, T = green

CSP as Search Problems

Incremental formulation:

states:
consistent assignments.

initial state:
empty assignment.

successor function:
assign any not yet assigned variable
s.t. the resulting assignment still is consistent.

goal test:
assignment is complete.

path cost:
constant cost 1 for each step.

Types of Variables & Constraints

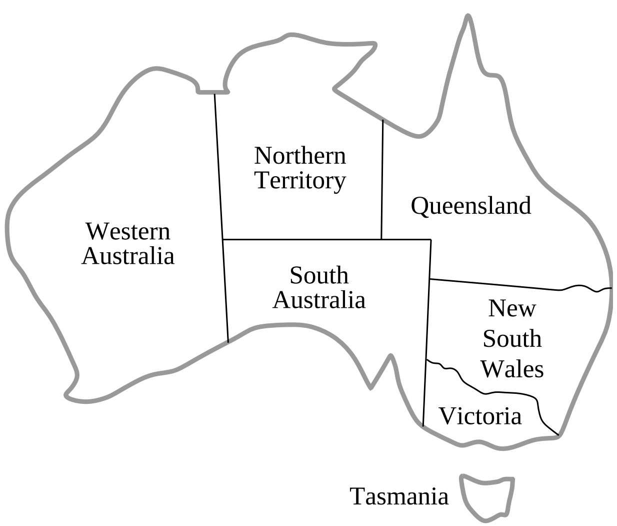
	finite domains	infinite domains
condition:	$\lvert \operatorname{dom} X_i \rvert \in \mathbb{N} \quad \forall i$	otherwise
example:	8-queens: $\lvert \operatorname{dom} Q_i \rvert = 8$. map coloring: $\lvert \operatorname{dom} X_i \rvert = 3$.	scheduling: $\operatorname{dom} X_i = \mathbb{N}$ (number of days from now)
special cases:	binary CSPs: $\lvert \operatorname{dom} X_i \rvert = 2$	integer domains: $\operatorname{dom} X_i = \mathbb{N}$ continuous domains: $\operatorname{dom} X_i = \mathbb{R}$ (or an interval)
constraints:	can be provided by enumeration, e.g., $(\text{WA}, \text{NT}) \in \{(r,g),(r,b),(g,r),(g,b),(b,r),(b,g)\}$	must be specified using a **constraint language**, e.g., linear constraints.

Binary Constraints

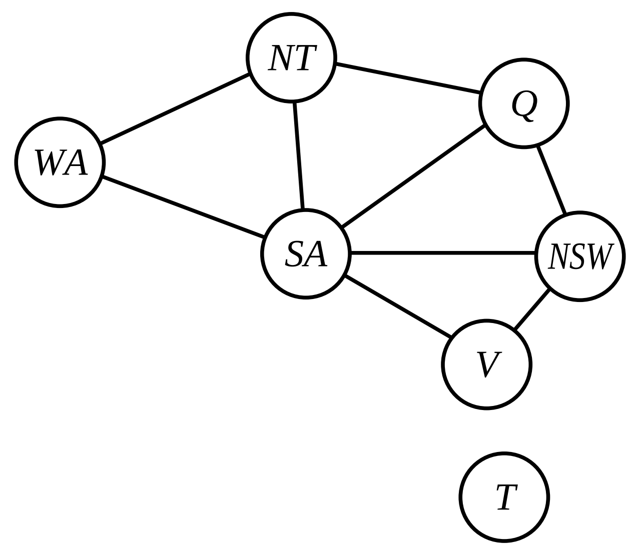
Constraints can be classified by the number $|\operatorname{var} C_j|$ of variables they depend on:

unary constraint: depends on a single variable X_i.
uninteresting: can be eliminated by inclusion in the domain $\operatorname{dom} X_i$.

binary constraint: depends on two variables X_i and X_j.
can be represented as a constraint graph.



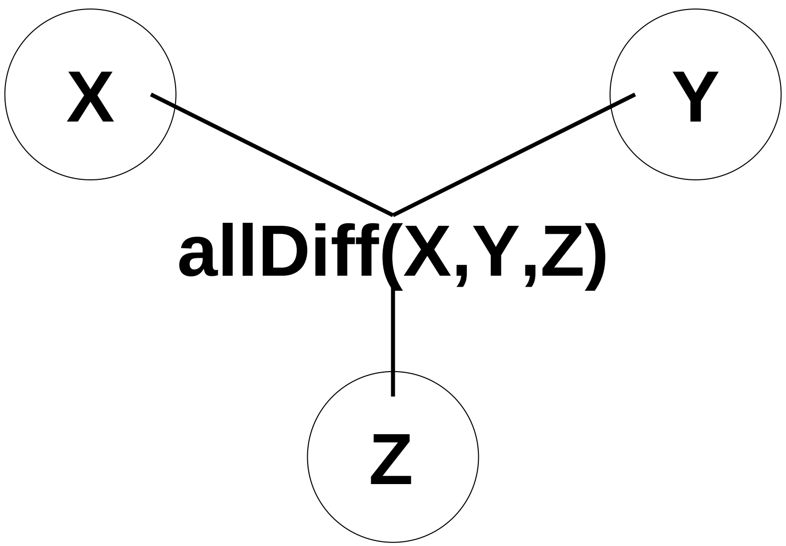
original map



constraint graph

n-ary Constraints

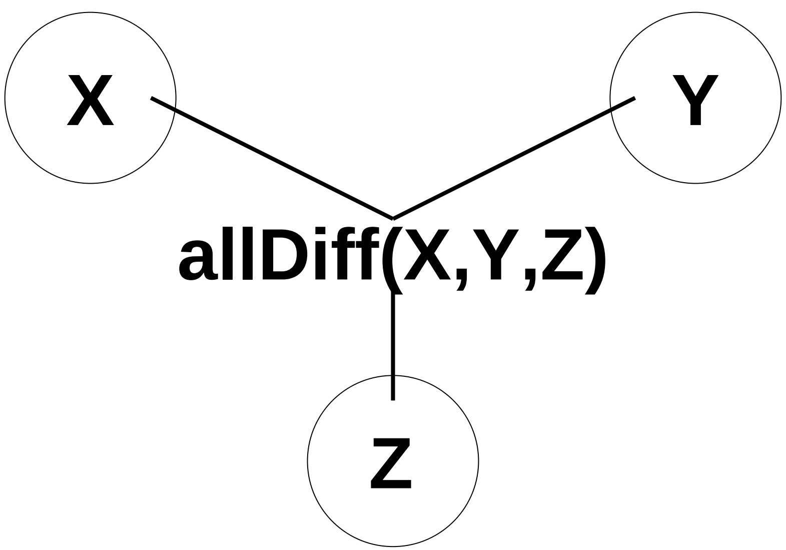
constraint of higher order / n**-ary constraint:** depends on more than two variables.
can be represented as a constraint hypergraph.



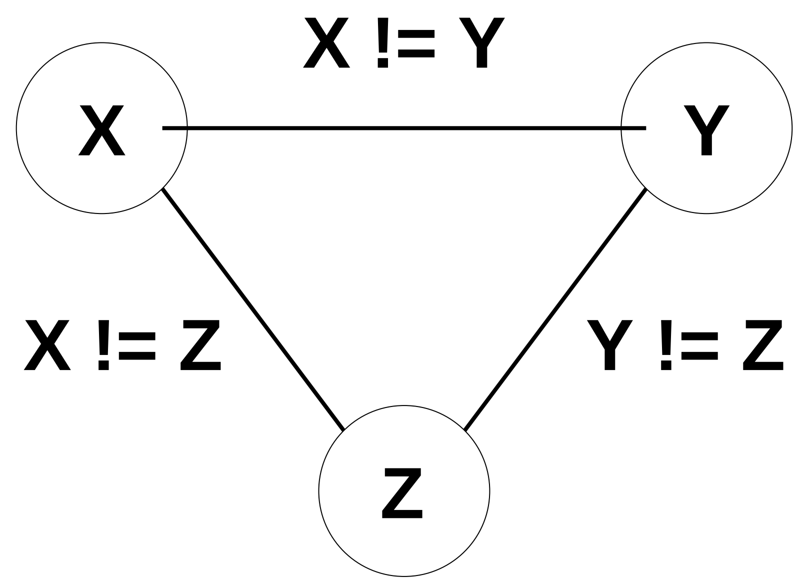
constraint hypergraph

n-ary Constraints

n-ary constraints sometimes can be reduced to binary constraints in a trivial way.



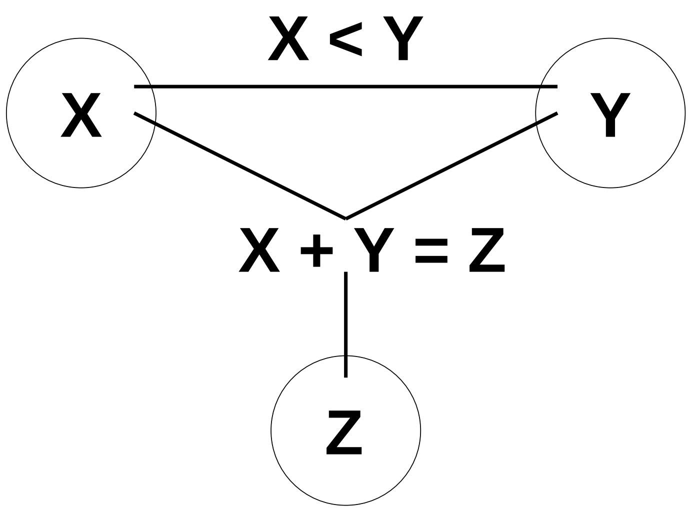
constraint hypergraph



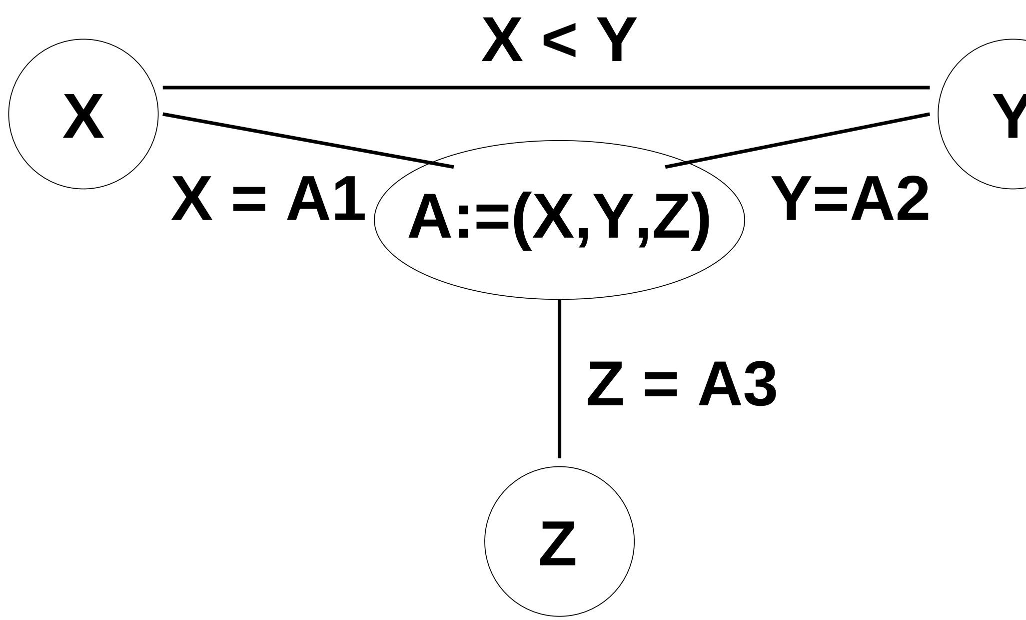
binarized constraint graph

n-ary Constraints

n-ary constraints always can be reduced to binary constraints by introducing additional **auxiliary variables** with the cartesian product of the original domains as new domain and the original n-ary constraint as unary constraint on the auxiliary variable.



constraint hypergraph



binarized constraint graph

Auxiliary Variables

Sometimes auxiliary variables also are necessary to represent a problem as CSP.

Example: cryptarithmetic puzzle.
Assign each letter a figure
s.t. the resulting arithmetic expression is true.

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

$$\begin{aligned} O + O &= R + 10X_1 \\ X_1 + W + W &= U + 10X_2 \\ X_2 + T + T &= O + 10X_3 \\ X_3 &= F \end{aligned}$$

Depth-First Search: Backtracking

Uninformed Depth-First search is called **backtracking** for CSPs.

backtracking(variables $\mathcal{X}$, constraints $\mathcal{C}$, assignment A) :
if $\mathcal{X} = \emptyset$ **return** A **fi**
$X :=$ choose($\mathcal{X}$)
$A' :=$ failure
for $v \in$ values$(X, A, \mathcal{C})$ **while** $A' =$ failure **do**
    $A' :=$ backtracking$(\mathcal{X} \setminus \{X\}, \mathcal{C}, A \cup \{X = v\})$
od
return A'

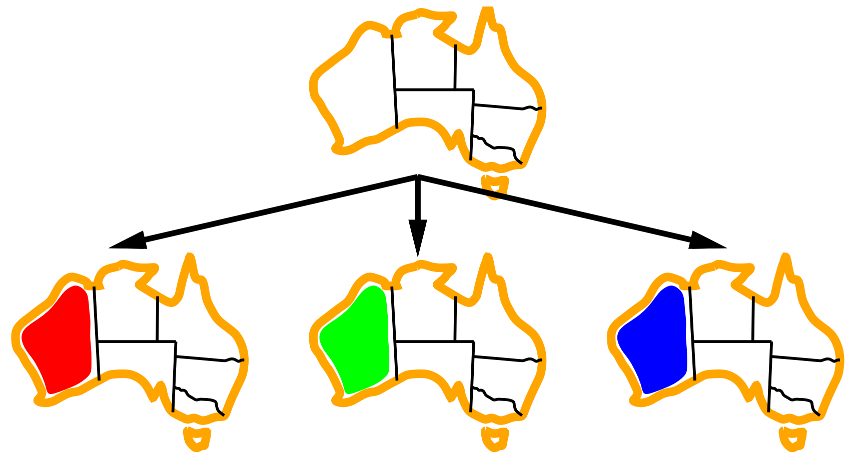
where

$$\text{values}(X, A, \mathcal{C}) := \{v \in \operatorname{dom} X \mid \forall C \in \mathcal{C} \text{ with } \operatorname{var} C \subseteq \operatorname{var} A \cup \{X\} : C(A, X = v) = \text{true}\}$$

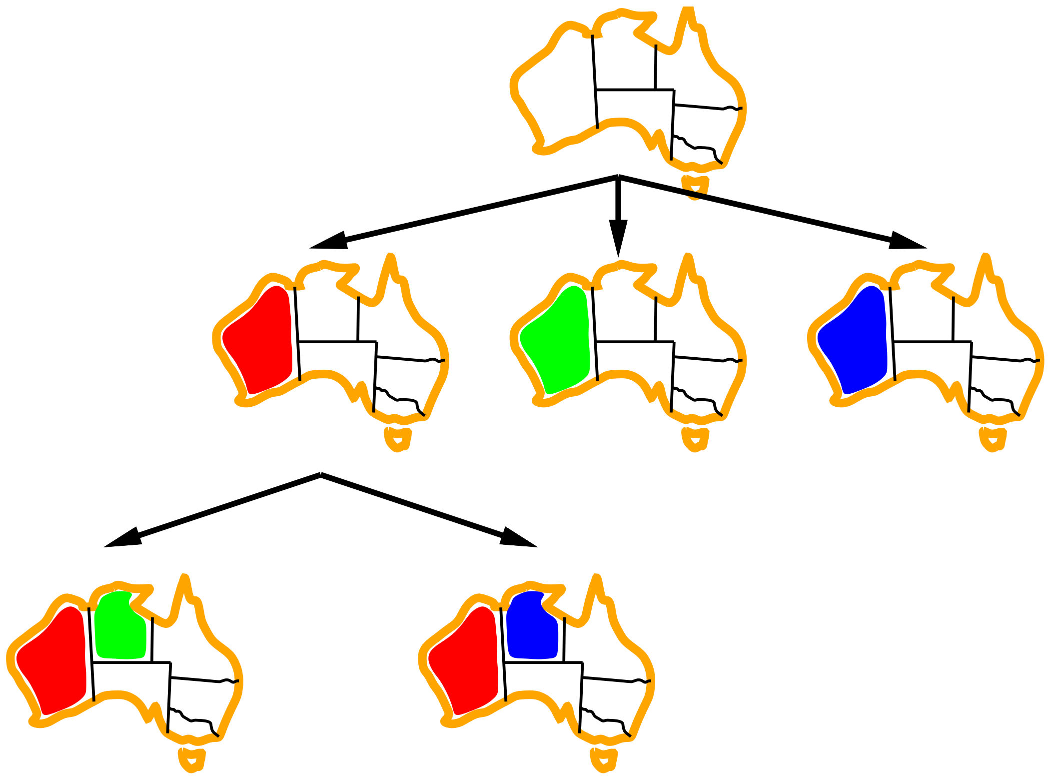
denotes the values for variable X consistent with assignment A for constraints $\mathcal{C}$.

Backtracking / Example

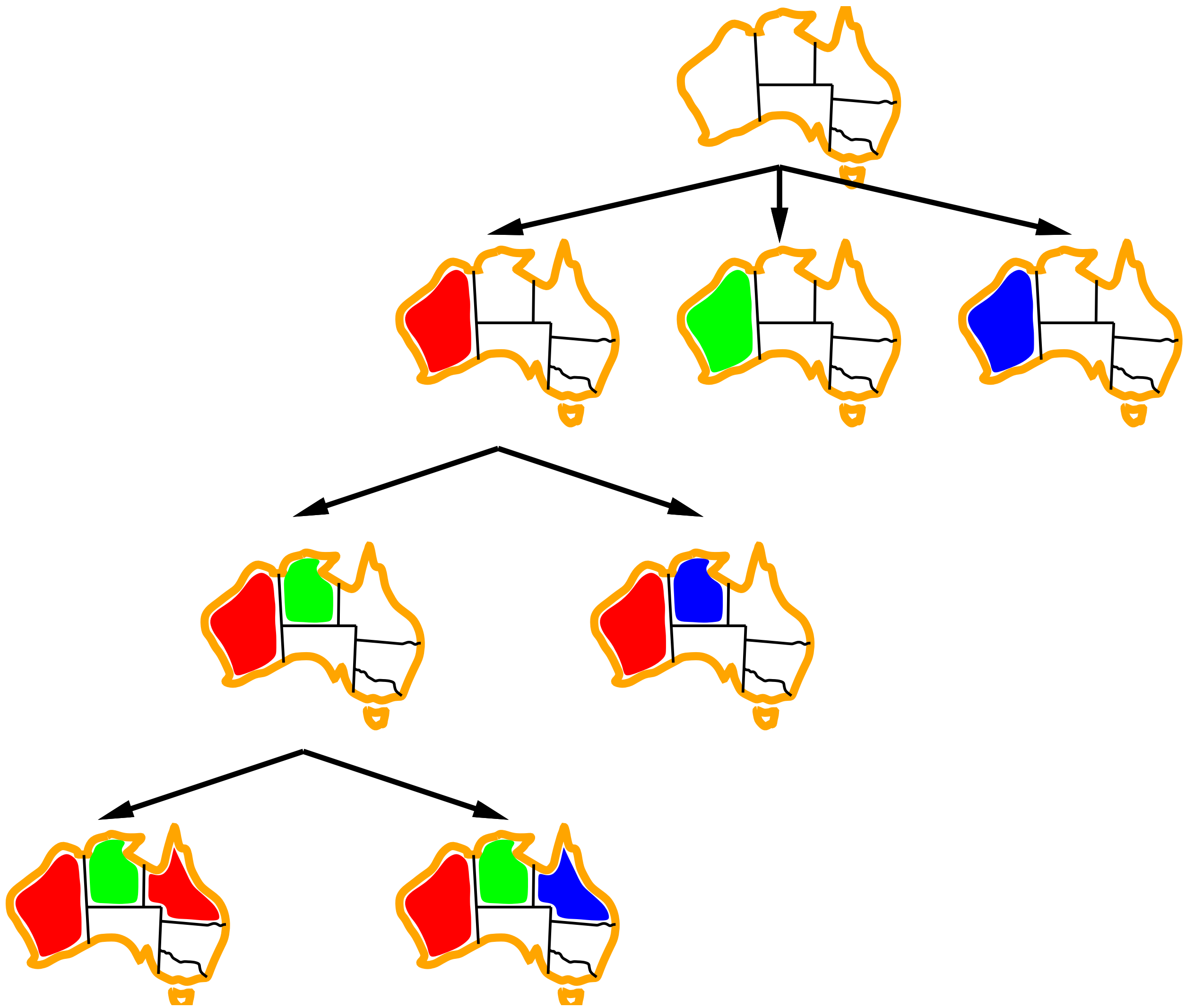
Backtracking / Example



Backtracking / Example



Backtracking / Example

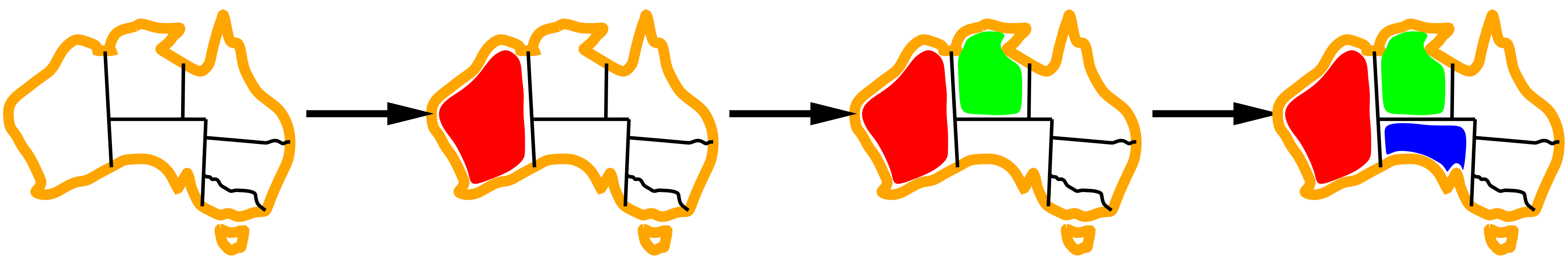


Variable Ordering / MRV

Which variable is selected in line 3 can be steered by heuristics:

minimum remaining values (MRV):
Select the variable with the smallest number of remaining choices:

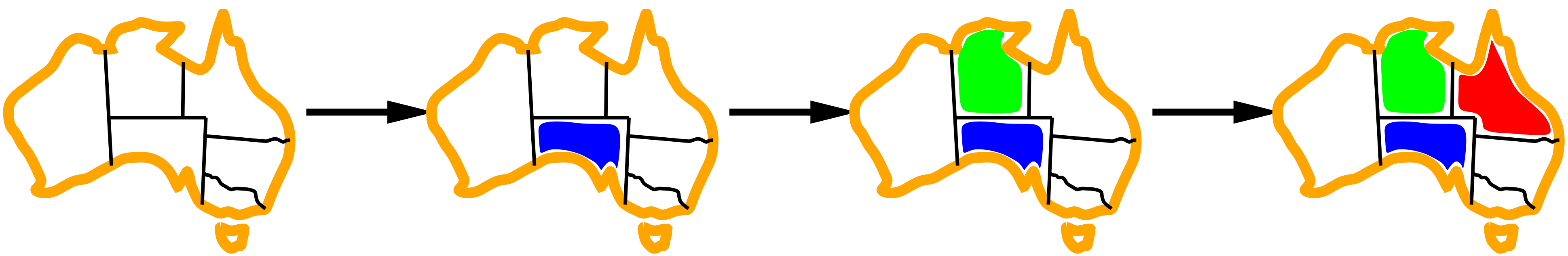
$$X := \text{argmin}_{X \in \mathcal{X}} |\text{values}(X, A, \mathcal{C})|$$



Variable Ordering / Degree Heuristics

degree heuristic:
Select the variable that is involed in the largest number of unresolved constraints:

$$X := \text{argmax}_{X \in \mathcal{X}} |\{C \in \mathcal{C} \mid X \in \text{var}\, C, \text{var}\, C \not\subseteq \text{var}\, A \cup \{X\}\}|$$



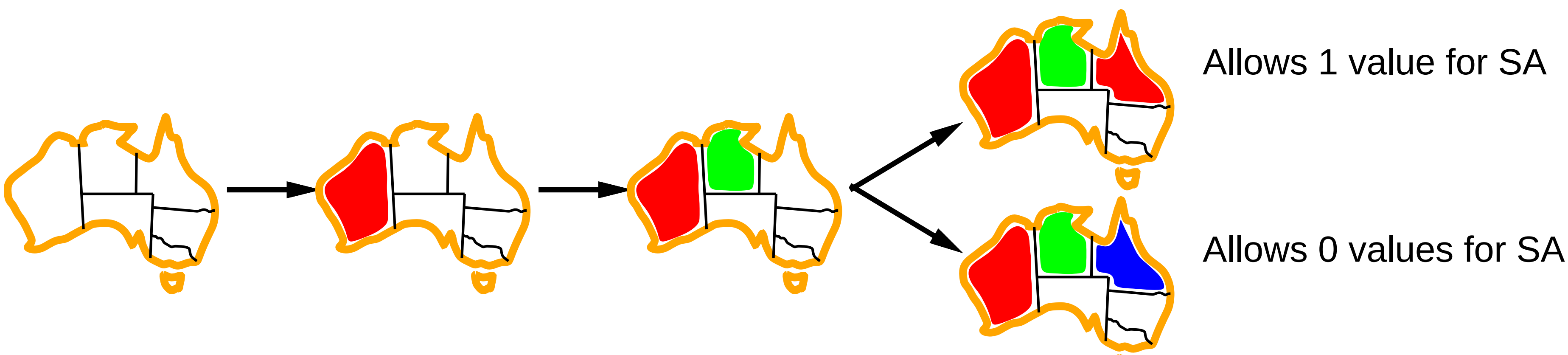
Usually one first applies MRV and breaks ties by degree heuristics.

Value Ordering

The order in which values for the selected variable are tried can also be steered by a heuristics:

least constraining value:
Order the values by descending number of choices for the remaining variables:

$$\sum_{Y \in \mathcal{X} \setminus \{X\}} |\text{values}(Y, A \cup \{X = v\}, \mathcal{C})|, \quad v \in \text{values}(X, A, \mathcal{C})$$

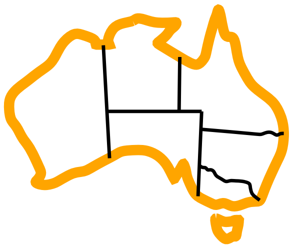


Forward Checking

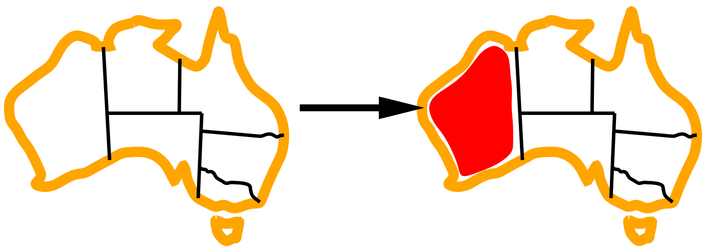
The minimum remaining values (MRV) heuristics can be implemented efficiently by keeping track of the remaining values values$(X, A, \mathcal{C})$ of all unassigned variables.
— This is called **forward checking**.

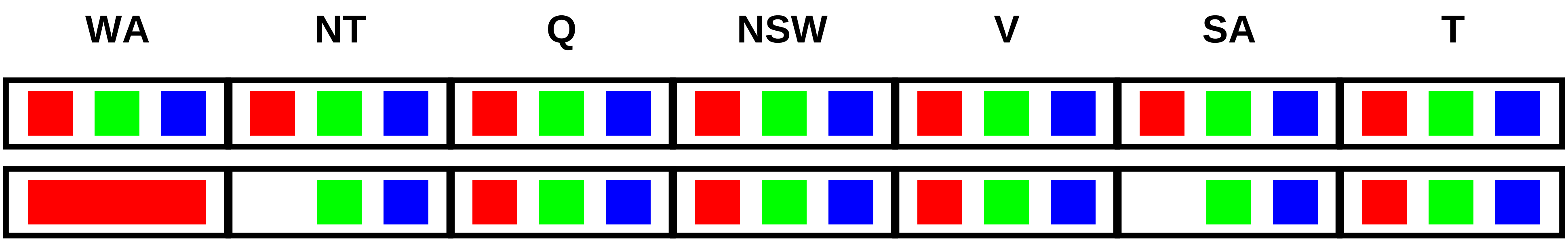
backtracking-fc(variables $\mathcal{X}$, (values$(X))_{X\in\mathcal{X}}$, constraints $\mathcal{C}$, assignment A) :
if $\mathcal{X} = \emptyset$ **return** A **fi**
$X := \operatorname{argmin}_{X\in\mathcal{X}} |\text{values}(X)|$
$A' :=$ failure
for $v \in$ values(X) **while** $A' =$ failure **do**
    illegal$(Y) := \{w \in \text{values}(Y) \mid \exists C \in \mathcal{C} : X, Y \in \operatorname{var} C, \operatorname{var} C \subseteq \operatorname{var} A \cup \{X, Y\},$
        $C(A, X = v, Y = w) = \text{false}\}, \quad \forall Y \in \mathcal{X} \setminus \{X\}$
    $A' :=$ backtracking$(\mathcal{X} \setminus \{X\}, (\text{values}(Y) \setminus \text{illegal}(Y))_{Y\in\mathcal{X}\setminus\{X\}}, \mathcal{C}, A \cup \{X = v\})$
od
return A'

Forward Checking

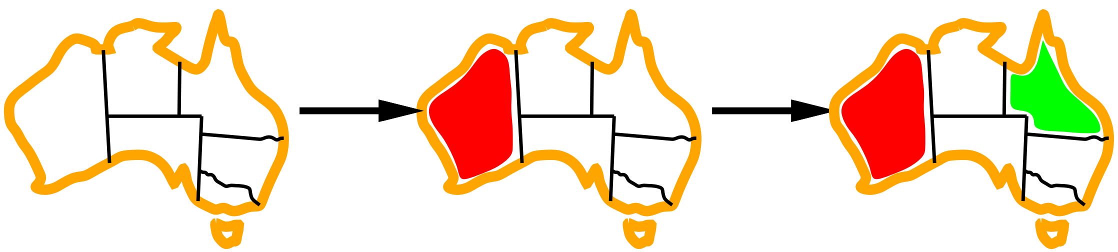


Forward Checking

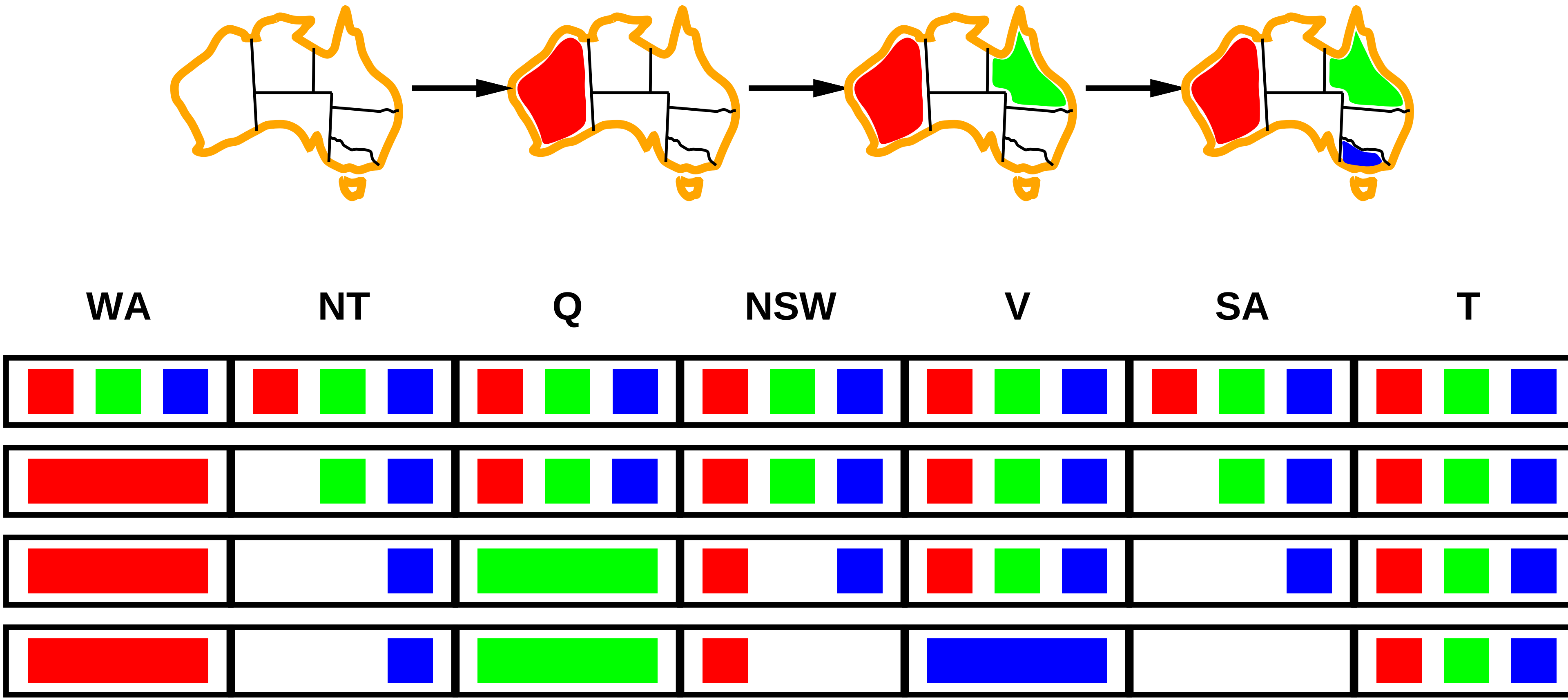




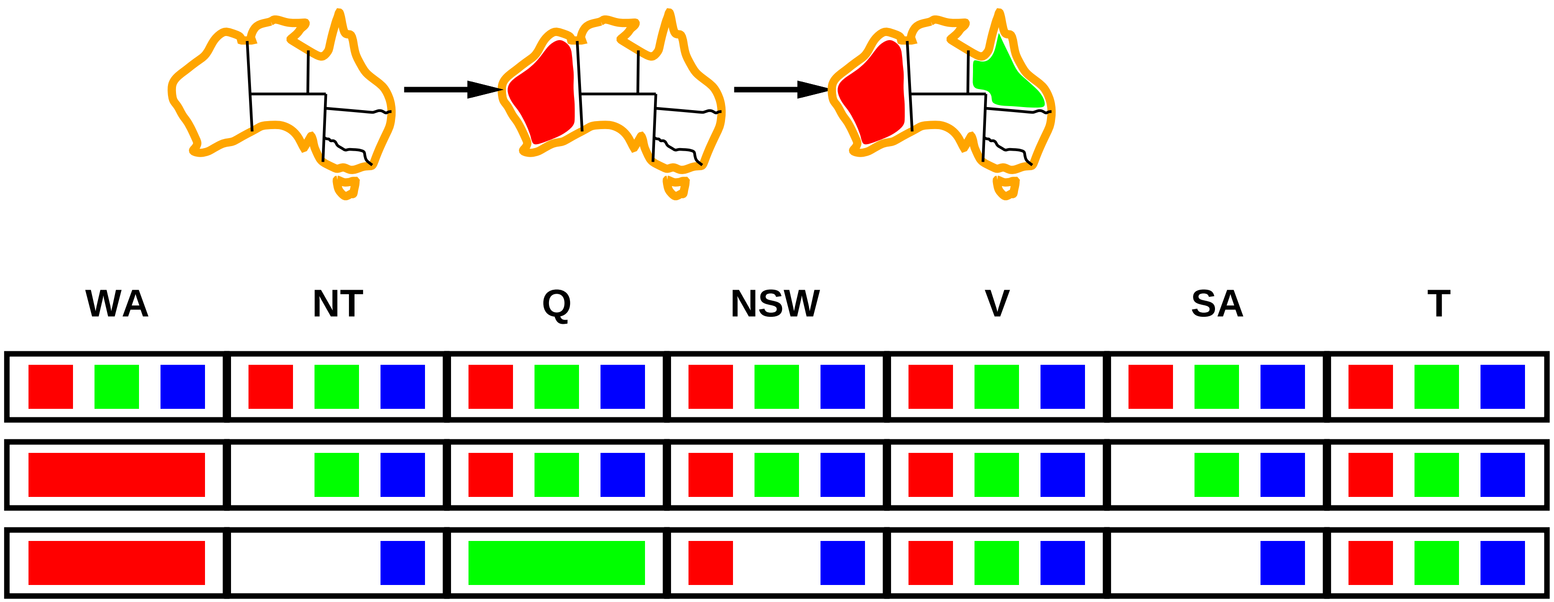
Forward Checking



Forward Checking



Constraint Propagation



Arc Consistency

One also could use a stronger consistency check: if

- there is for some unassigned variable X a possible value v,
- there is a constraint C linking X to another unassigned variable Y, and
- setting $X = v$ would rule out all remaining values for Y via C,

then we can remove v as possible value for X.

Example:

$$\text{values}(\text{SA}) = \{b\}, \quad \text{values}(\text{NSW}) = \{r, b\}, \quad C : \text{NSW} \neq \text{SA}$$

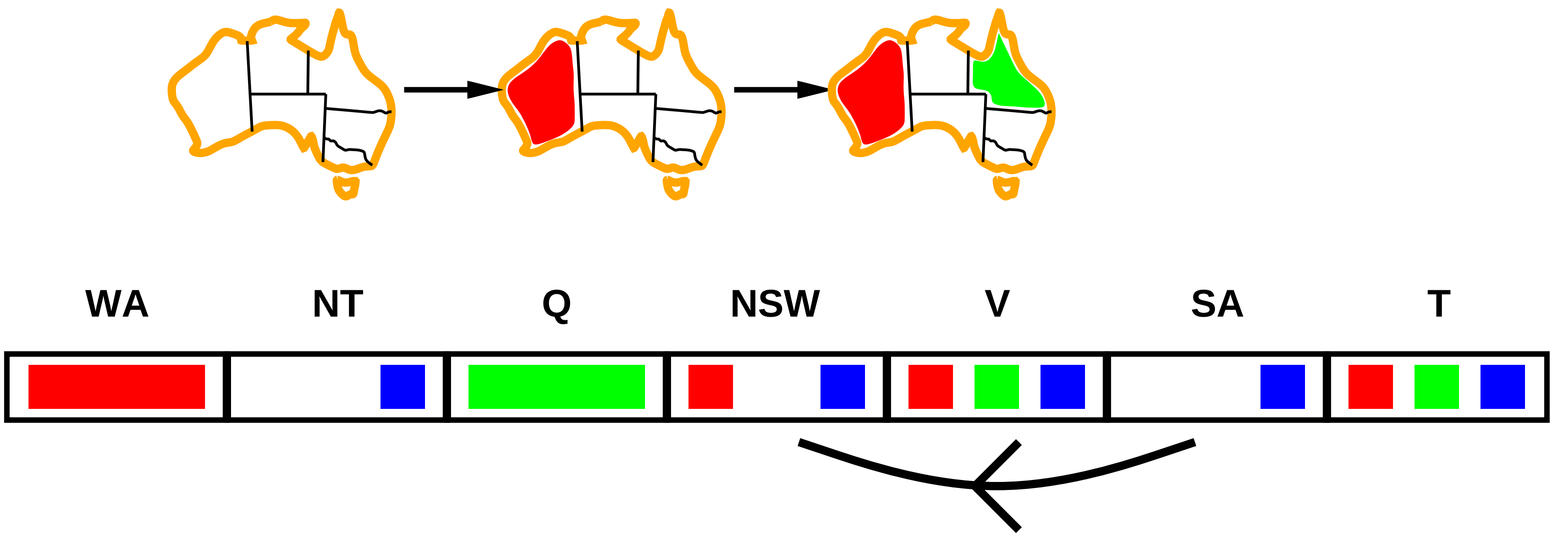
NSW $= b$ is not possible as C would lead to values(SA) $= \emptyset$.

Removing such a value may lead to other inconsistent arcs, thus, has to be done repeatedly.

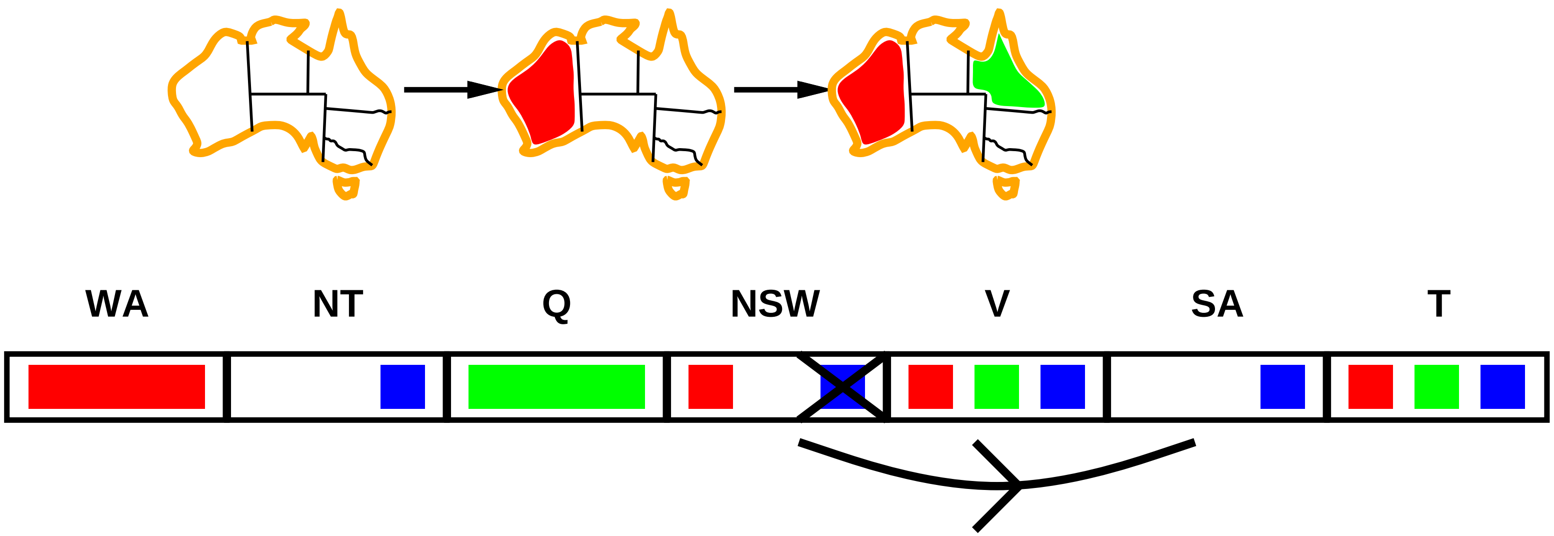
Arc Consistency

arc-consistency(variables $\mathcal{X}$, $(\text{values}(X))_{X\in\mathcal{X}}$, constraints $\mathcal{C}$) :
arcs := $((X, Y, C) \in \mathcal{X}^2 \times \mathcal{C} \mid \text{var}\, C = \{X, Y\})$ in any order
while arcs $\neq \emptyset$ **do**
    (X, Y, C) := remove-first(arcs)
    illegal := $\{v \in \text{values}(X) \mid \forall w \in \text{values}(Y) : C(X = v, Y = w) = \text{false}\}$
    if illegal $\neq \emptyset$
        values(X) := values$(X) \setminus$ illegal
        append(arcs, $((Y', X', C') \in \mathcal{X}^2 \times \mathcal{C} \mid X' = X, Y' \neq Y, \text{var}\, C' = \{X', Y'\})$)
    fi
od
return $(\text{values}(X))_{X\in\mathcal{X}}$

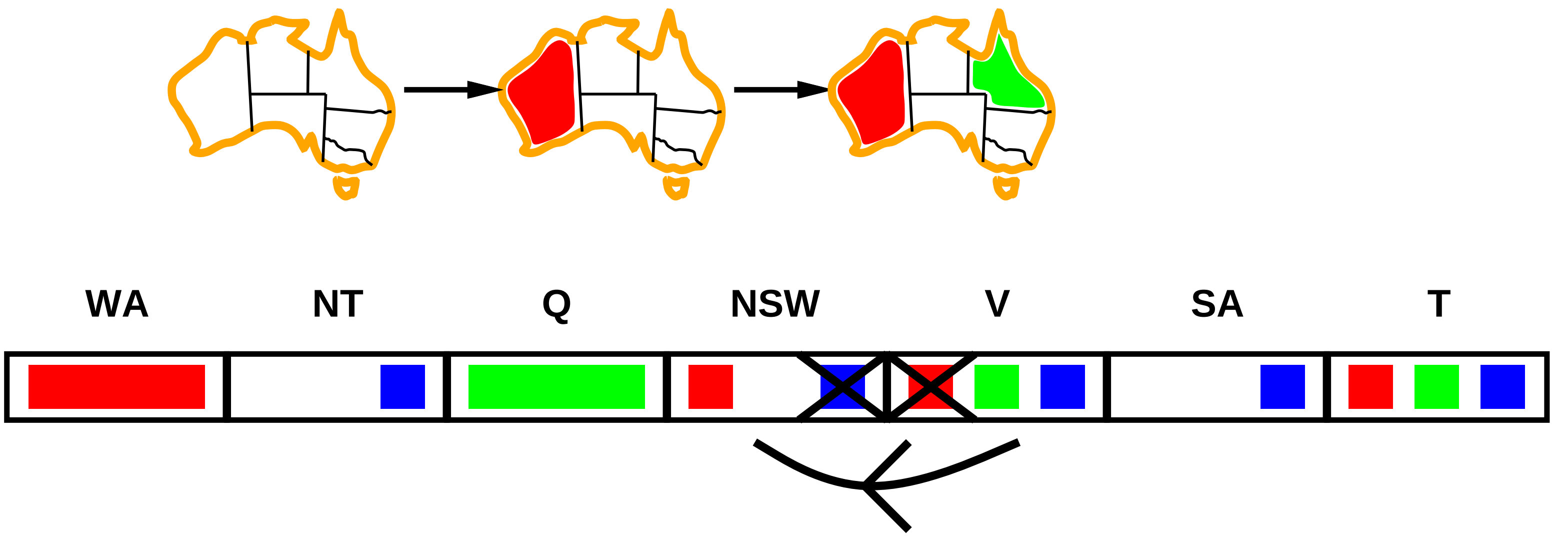
Arc Consistency



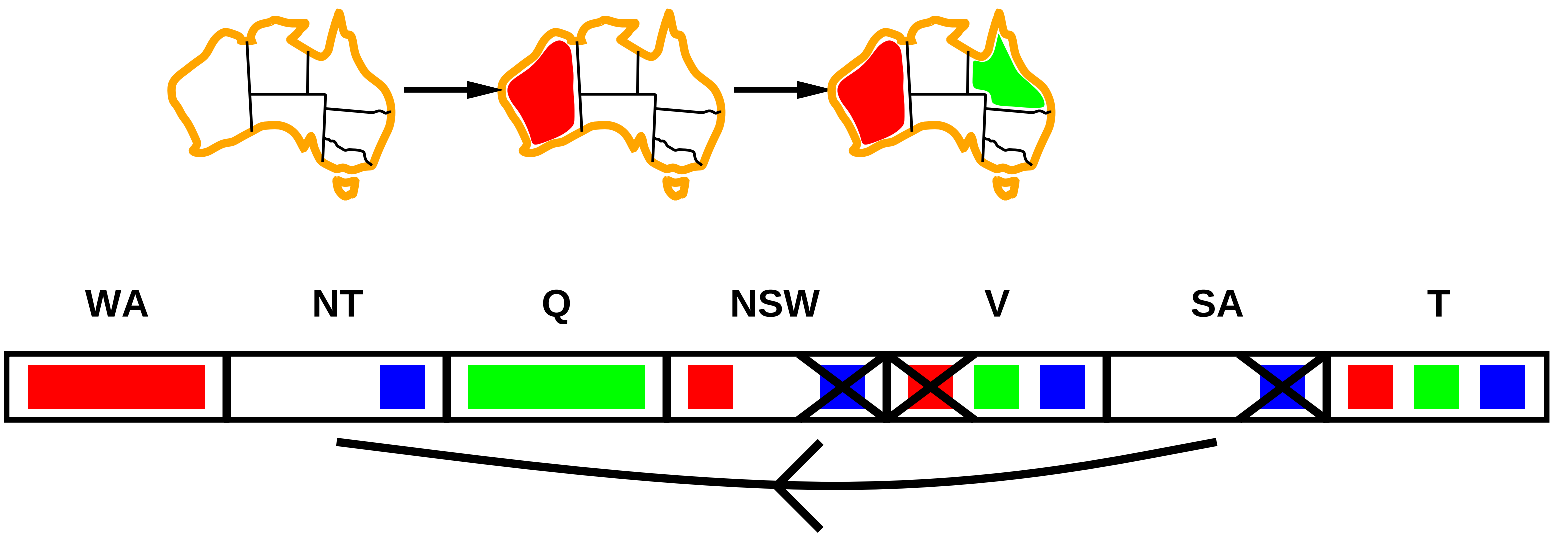
Arc Consistency



Arc Consistency



Arc Consistency



k-consistency

k-consistency:
any consistent assignment of any $k-1$ variables can be extended to a consistent assignment of k variables with any k-th variable.

1-consistency: node consistency
same as forward checking.

2-consistency: arc consistency

3-consistency: path consistency

strong k-consistent: 1-consistent and 2-consistent and . . . and k-consistent.

strong n-consistency (where n is the number of variables) renders a CSP trivial:
select a value for X_1, compute the remaining values for the other variables, then pick on for X_2 etc. — strong n-consistency guarantees that there is no step where backtracking is necessary.

min conflicts

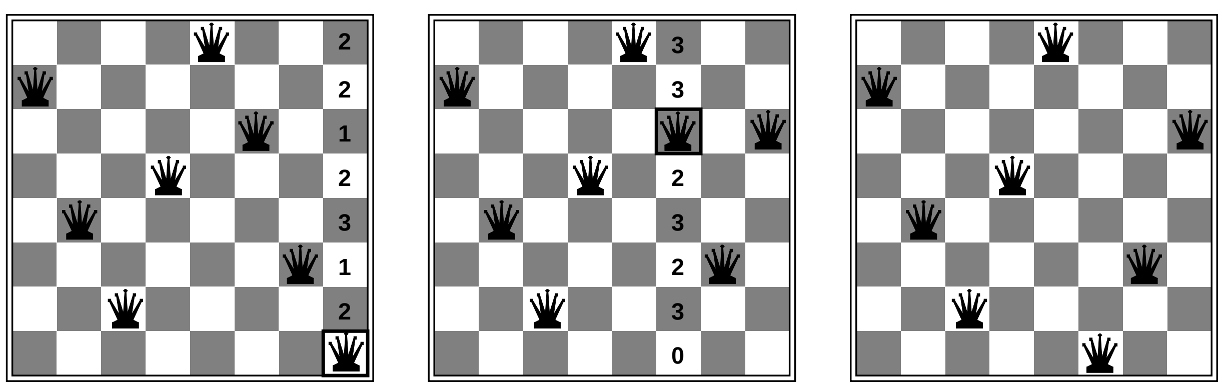
sort of greedy local search:

states: complete assignments

neighborhood: re-assigning a (randomly picked) conflicting variable

goal: no conflicts

min-conflicts(variables $\mathcal{X}$, constraints $\mathcal{C}$) :
$A :=$ random complete assignment for $\mathcal{X}$
for $i := 1 \ldots$ maxsteps **while** $\exists C \in \mathcal{C} : C(A) = \text{false}$ **do**
  $X := \text{random}(\{X \in \mathcal{X} \mid \exists C \in \mathcal{C} : C(A) = \text{false and } X \in \text{var}\, C\})$
  $v := \text{argmin}_{v \in \text{dom}\, X} |\{C \in \mathcal{C} \mid C(A, X = v) = \text{false}, X \in \text{var}\, C\}|$
  $A|_X := v$
od
return A, if $\forall C \in \mathcal{C} : C(A) = \text{true}$, failure else

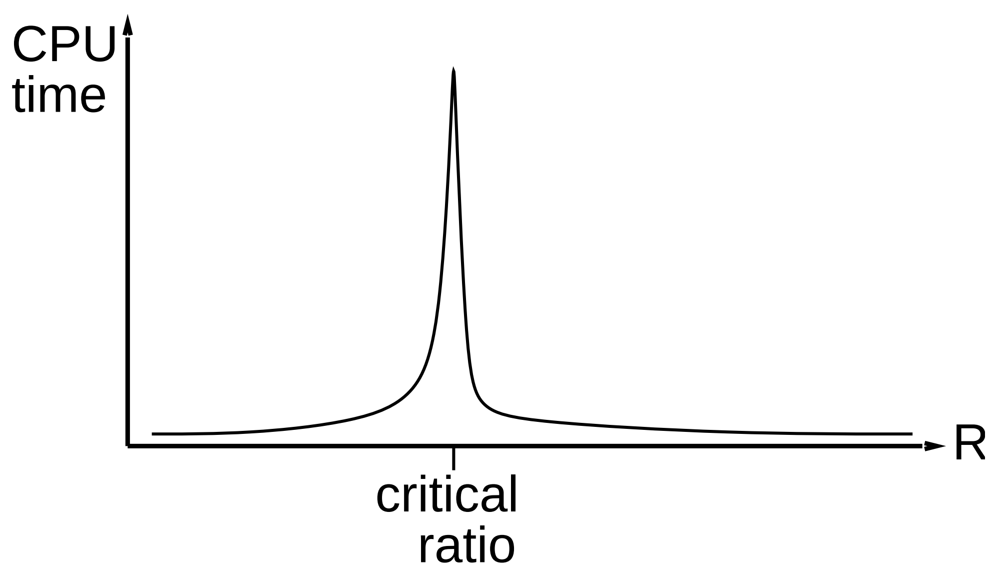


min conflicts / performance

min conflicts finds solution for n-queens problem very quickly even for very large n, e.g., $n = 10,000,000$ (starting from a random initial state).

min conflicts also can solve large randomly-generated CSPs very quickly
except in a narrow range of the constraints / variables ratio

$$R := \frac{\text{number of constraints}}{\text{number of variables}}$$



1. Constraint Satisfaction Problems

2. Backtracking Search

3. Local Search

4. The Structure of Problems

Connected Components / Graphs

Let $G := (V, E)$ be an undirected graph.
A sequence $p = (p_1, \ldots, p_n) \in V^*$ of vertices is called **path** of G if

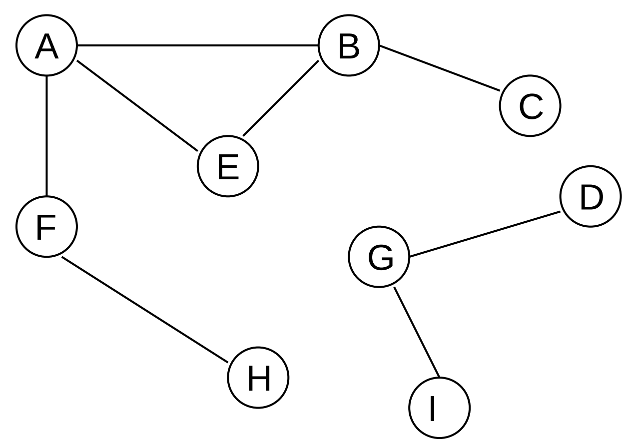
$$(p_i, p_{i+1}) \in E \quad \text{for } i = 1 \ldots, n-1$$

G^* denotes the set of paths on G.

$x, y \in V$ are called **connected** if there is a path in G between x and y,
i.e., it exists $p \in G^*$ with $p_1 = x$ and $p_{|p|} = y$.

G is called **connected** if all pairs of vertices are connected.

A maximal connected subgraph $G' := (V', E')$ of G is called **connection component of** G.



Connected Components / Graphs

Let $G := (V, E)$ be an undirected graph.
A sequence $p = (p_1, \ldots, p_n) \in V^*$ of vertices is called **path** of G if

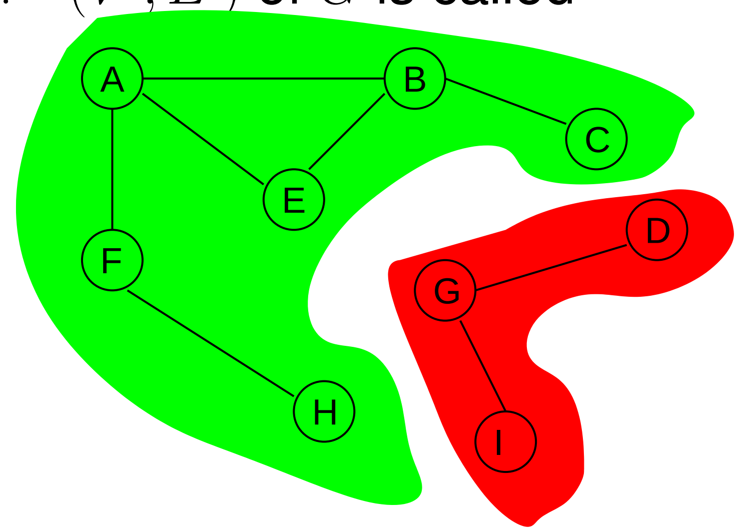
$$(p_i, p_{i+1}) \in E \quad \text{for } i = 1 \ldots, n-1$$

G^* denotes the set of paths on G.

$x, y \in V$ are called **connected** if there is a path in G between x and y,
i.e., it exists $p \in G^*$ with $p_1 = x$ and $p_{|p|} = y$.

G is called **connected** if all pairs of vertices are connected.

A maximal connected subgraph $G' := (V', E')$ of G is called **connection component of** G.



Connected Components / Hypergraphs

Let $G := (V, E)$ be a hypergraph, i.e., $E \subseteq \mathcal{P}(V)$.
A sequence $p = (p_1, \ldots, p_n) \in E^*$ of edges is called **path** of G if

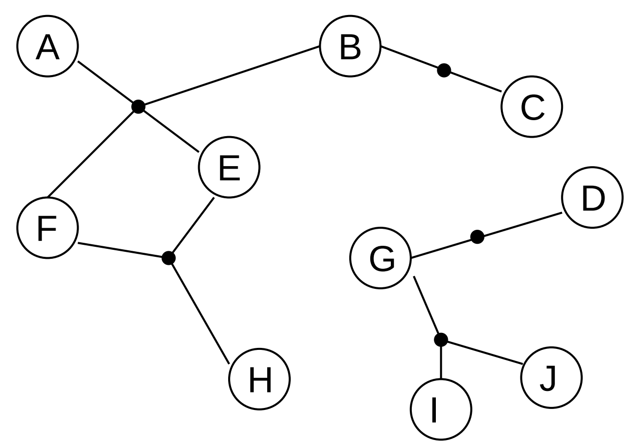
$$p_i \cap p_{i+1} \neq \emptyset \quad \text{for } i = 1 \ldots, n-1$$

G^* denotes the set of paths on G.

$x, y \in V$ are called **connected** if there is a path in G between x and y,
i.e., it exists $p \in G^*$ with $x \in p_1$ and $y \in p_{|p|}$.

G is called **connected** if all pairs of vertices are connected.

A maximal connected subgraph $G' := (V', E')$ of G is called **connection component of** G.



Connected Components / Hypergraphs

Let $G := (V, E)$ be a hypergraph, i.e., $E \subseteq \mathcal{P}(V)$.
A sequence $p = (p_1, \ldots, p_n) \in E^*$ of edges is called **path** of G if

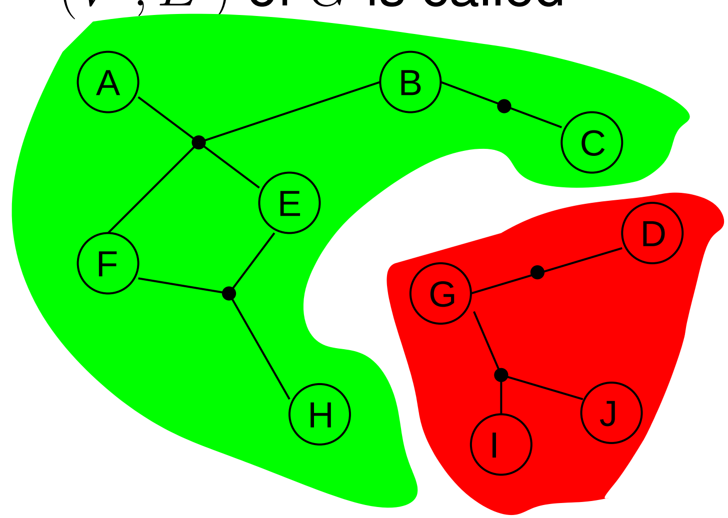
$$p_i \cap p_{i+1} \neq \emptyset \quad \text{for } i = 1 \ldots, n-1$$

G^* denotes the set of paths on G.

$x, y \in V$ are called **connected** if there is a path in G between x and y,
i.e., it exists $p \in G^*$ with $x \in p_1$ and $y \in p_{|p|}$.

G is called **connected** if all pairs of vertices are connected.

A maximal connected subgraph $G' := (V', E')$ of G is called **connection component of** G.



Independent Subproblems

Let $(\mathcal{X}, \mathcal{C})$ be a constraint satisfaction problem.
The CSP $(\mathcal{X}', \mathcal{C}')$ with $\mathcal{X}' \subseteq \mathcal{X}$ and

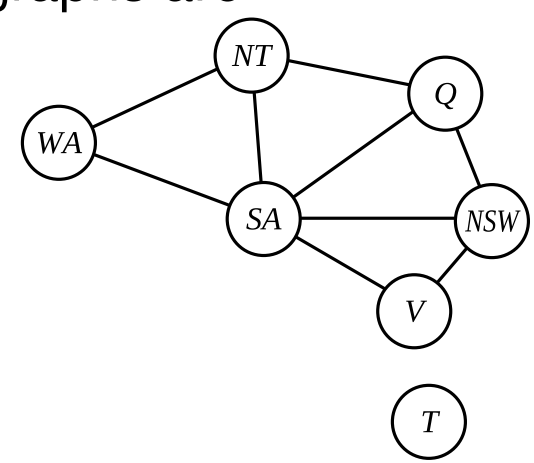
$$\mathcal{C}' := \{C \in \mathcal{C} \mid \operatorname{var} C \subseteq \mathcal{X}'\}$$

is called **subproblem of** $(\mathcal{X}, \mathcal{C})$ **on the variables** $\mathcal{X}'$.

Two subproblems on the variables $\mathcal{X}'_1$ and $\mathcal{X}'_2$ are called **independent** if there is no joining constraint, i.e., no $C \in \mathcal{C}$ with

$$\operatorname{var} C \cap \mathcal{X}'_1 \neq \emptyset \text{ and } \operatorname{var} C \cap \mathcal{X}'_2 \neq \emptyset$$

(and thus $\mathcal{X}'_1 \cap \mathcal{X}'_2 = \emptyset$).
I.e., if the respective constraint sub-hypergraphs are unconnected.



Independent Subproblems

Consistent assignments of independent subproblems can be joined to consistent assignments of the whole problem.

The other way around:
if a probem decomposes into independent subproblems
we can solve each one separately
and joint the subproblem solutions afterwards.

Tree Constraint Graphs

The next simple case:
If the constraint graph is a tree,
there is a linear-time algorithm to solve the CSP:

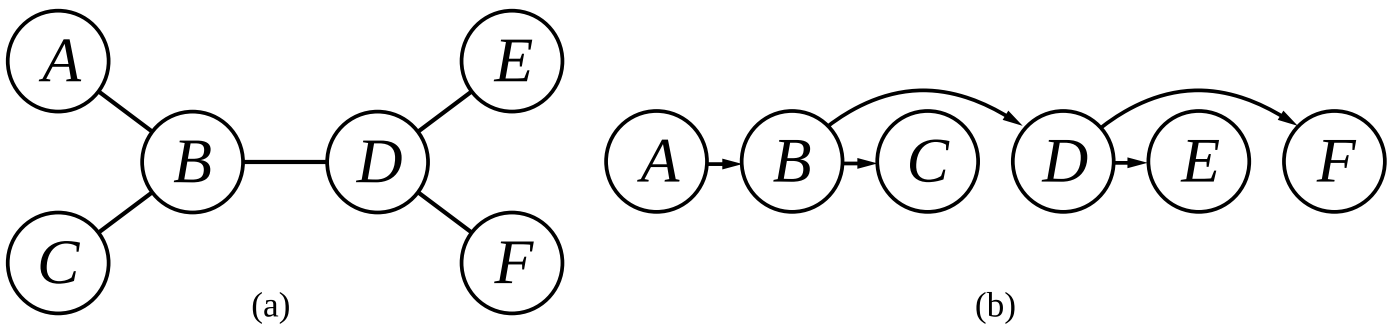
1. choose any vertex as the root of the tree,

2. order the variables from root to leaves
   s.t. parents precede their children in the ordering.
   (topological ordering)
   Denote variables by $X_{(1)}, X_{(2)}, \ldots, X_{(n)}$.

3. For $i = n$ down to 2:
   apply arc consistency to the edge $(\text{parent}(X_{(i)}), X_{(i)})$
   i.e., eventually remove values from dom parent$(X_{(i)})$.

4. For $i = 1$ to n:
   choose a value for $X_{(i)}$ consistent with the value already
   choosen for parent$(X_{(i)})$.

Tree Constraint Graphs



General Constraint Graphs

Idea: try to reduce problem to constraint trees.

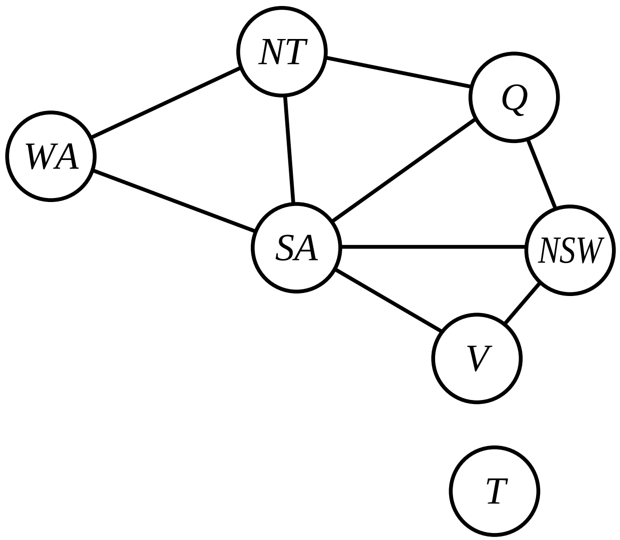
Approach 1: **cycle cutset**
remove some vertices s.t. the remaining vertices form a tree.

for binary CSPs:

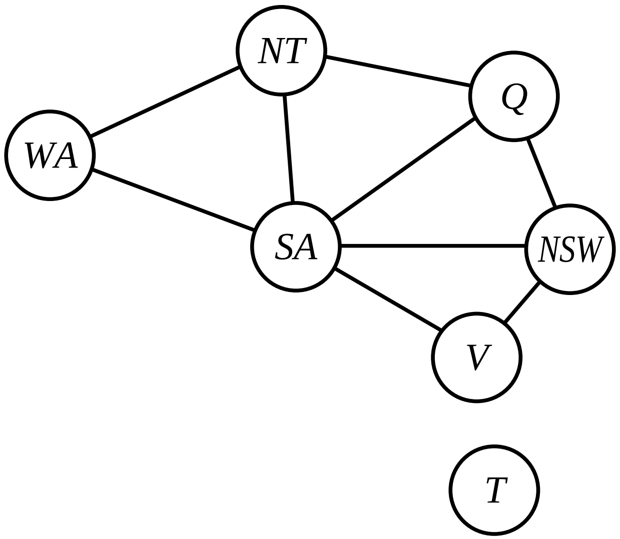
1. find a subset $S \subseteq \mathcal{X}'$ of variables
   s.t. the constraint graph of the subproblem on $\mathcal{X} \setminus S$ becomes a tree.

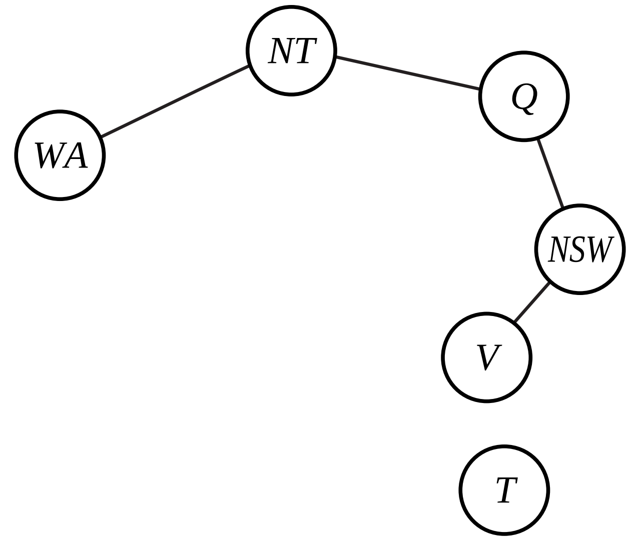
2. for each consistent assignment A on S:
   (a) remove from the domains of $\mathcal{X} \setminus S$ all values not consistent with A,
   (b) search for a solution of the remaining CSP.
   if there is one, an overall solution has been found.

General Constraint Graphs / Cycle cutset



General Constraint Graphs / Cycle cutset





The smaller the cutset, the better.

Finding the smallest cutset is NP-hard.

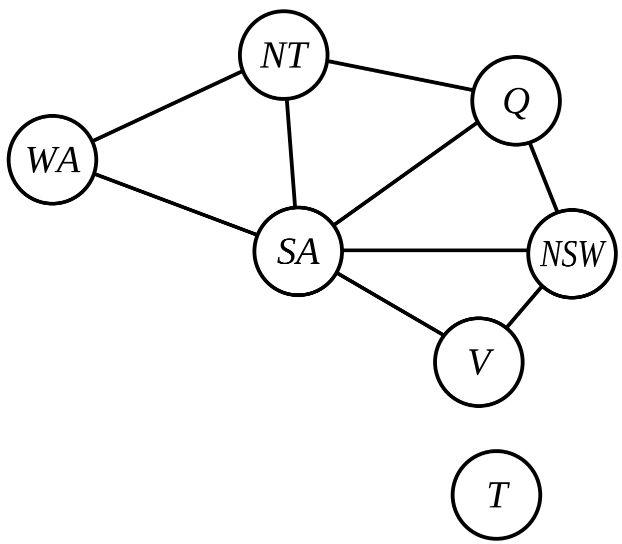
General Constraint Graphs / Tree Decompositions

Approach 2: **tree decomposition**
decompose the constraint graph in overlapping subgraphs
s.t. the overlapping structure forms a tree

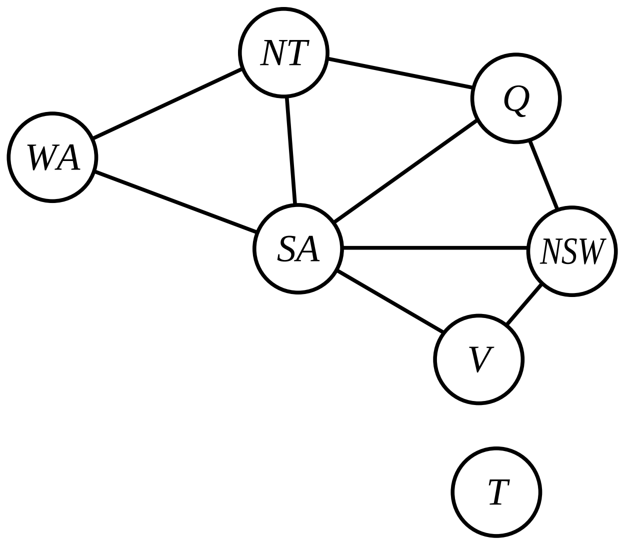
Tree decomposition $(\mathcal{X}_i)_{i=1,\ldots,m}$:

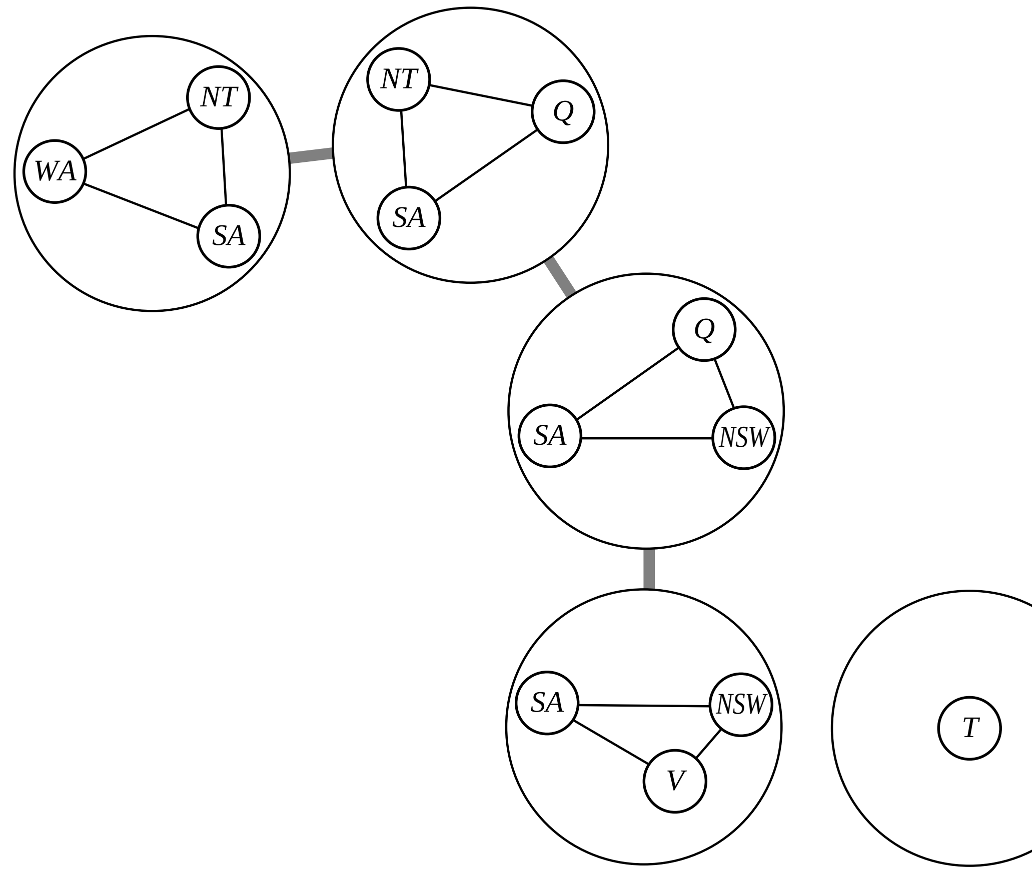
1. each vertex appears in at least one subgraph.
2. each edge appears in at least one subgraph.
3. if a vertex appears in two subgraphs, it must appear in every subgraph along the path connecting those two vertices.

General Constraint Graphs / Tree Decompositions



General Constraint Graphs / Tree Decompositions





General Constraint Graphs / Tree Decompositions

To solve the CSP:
view each subgraph as a new variable
and apply the algorithm for trees sketched earlier.

Example:
(WA,SA,NT) = (r,b,g) $\Rightarrow$ (SA,NT,Q) = (b,g,r)

In general, many tree decompositions possible.

The **treewidth** of a tree decomposition is the size of the largest subgraph minus 1.

The smaller the treewidth, the better.

Finding the tree decomposition with minimal treewidth is NP-hard.

Summary

- CSPs allow to describe problems by **variables** and **constraints** between them.
- Depth-first search assigning one variable a time (called **backtracking**) can be used to solve CSPs.
- Heuristics for choosing the next variable to assign (**MRV**; **degree heuristics**) and for ordering the values (**least constraining value**) can accelerate backtracking.
- MRV can be efficiently implemented keeping book of the remaining values for each unassigned variable (**forward checking**).
- More complex methods of constraint propagation (such as **arc consistency**) can be used to lower the risk of having to backtrack.
- Local search (**min conflicts**) can be used to solve CSPs quickly.
- **Tree-structured CSPs** can be solved in linear time.