I can’t get no...

Constraint Satisfaction

Professor Marie Roch
Chapter 6, Russell & Norvig
(skip 6.3.4, 6.5)
Constraint satisfaction problems (CSP)

Solutions with caveats

Example: Find a way to take classes such that I graduate in four years

• prerequisites
• course availability
• funding
Constraint satisfaction problems (CSP)

• To date, states were
  • atomic – didn’t care about internal representation except with respect to analyzing for goal/heuristic
  • mutated by actions that produced a new atomic state

• Factored representations
  • states have internal structure
  • structure can be manipulated
  • constraints relate different parts of the structure to one another and provide legal/illegal configurations
CSP Definition

Problem = \{X, D, C\}

- \(X\) – Set of variables
  
  \(X = \{X_1, X_2, \ldots, X_n\}\)

- \(D\) – Set of domains such that
  
  \(D = \{D_1, D_2, \ldots, D_n\}\)

  \(X_i = x_i\) where \(x_i \in D_i\)

- \(C\) – Set of constraints such that
  
  \(C = \{C_1, C_2, \ldots, C_m\}\)

  \(C_i = \langle (Variables), relationship(variables) \rangle\)

  Example: \(\langle (X_2, X_5), X_2 \neq X_5 \rangle\)
CSP example: map coloring

- Color territories on a map using 3 colors such that no two colors are adjacent

One possible solution for colors: orange, blue, and green

Note: 4 colors are sufficient to color any map
Map coloring

• Graph representation

• Variables
  \(X = \{\text{WA, NT, SA, Q, NSW, V, T}\}\)

• All variables have the same domain
  \(D_i = \{\text{red, green, blue}\}\)

• Constraint set
  \(C = \{\text{SA \neq WA, SA \neq NT, SA \neq Q, SA \neq NSW, SA \neq V, WA \neq NT, NT \neq Q, Q \neq NSW, NSW \neq V}\}\)
  or \(\{\text{adjacent}(t_a,t_b) \rightarrow t_a \neq t_b\}\)
Scheduling example

Partial auto assembly

• Install front and rear axels (10 min each)
• Install four wheels (1 min each)
• Install nuts on wheels (2 min each wheel)
• Attach hubcap (1 min each)
• Inspect

Variable set $X$:

$$X = \left\{ Axle_F, Axle_B, Wheel_{RF}, Wheel_{LF}, Wheel_{RB}, Wheel_{LB}, Nuts_{RF}, Nuts_{LF}, Nuts_{RB}, Nuts_{LB}, Cap_{RF}, Cap_{LF}, Cap_{RB}, Cap_{LB}, \text{Inspect} \right\}$$
Constraint types

• Domain values
  • Time at which task begins \{0, 1, 2, \ldots\}

• Precedence constraints
  • Suppose it takes 10 minutes to install axles.
  • We can ensure that front wheels are not started before axle assembly is completed:

\[
Axle_F + 10 \leq Wheel_{RF}
\]
\[
Axle_F + 10 \leq Wheel_{LF}
\]

• Disjunctive constraints – e.g. doohickey needed to assemble axle, but only have one

\[
Axle_F + 10 \leq Axle_B \text{ or } Axle_B + 10 \leq Axle_F
\]
Constraint types

• Unary – single variable
  \[ Z \leq 10 \]

• Binary – between two variables
  \[ Z^2 > Y \]

• Global – constraints with 3+ variables
  can be reduced to multiple binary/unary constraints

\[ X \leq Y \leq Z \rightarrow X \leq Y \text{ and } Y \leq Z \]

all\text{diff} (W, X, Y, Z) \rightarrow W \neq X, W \neq Y, W \neq Z, X \neq Y, ...

Note: Global constraints do not have to involve all variables
Constraint graphs


\[ T \quad W \quad O \]
\[ + \quad T \quad W \quad O \]
\[ F \quad O \quad U \quad R \]

**CSP specification**
- \( X = \{ F, T, U, W, R, O, C_1, C_2, C_3 \} \)
- \( D = \{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 \} \)
- \( C = \{
\begin{align*}
O + O &= R + 10 \quad C_1 \\
C_1 + W + W &= U + 10 \quad C_2 \\
C_2 + T + T &= O + 10 \quad C_3 \\
C_3 &= F \\
\text{Alldiff}(F, T, U, W, R, O)
\end{align*}
\} \)

\( \text{Alldiff}(X_1, X_2, \ldots, X_i) \rightarrow \forall j, k: j \neq k \text{ and } 1 \leq j, k \leq i, x_j \neq x_k \)

**Cryptoarithmetic puzzle**
Find a different digit for each letter such that substitution results in a valid equation.

\( C_i 's \) are auxiliary variables for carry digits
Constraint hypergraphs

- $O + O = R + 10C_1$
- $C_1 + W + W = U + 10C_2$
- $C_2 + T + T = O + C_3$
- $C_3 = F$
- `Alldiff(F,T,U,W,R,O)`
Binarization of constraints

• Convert n-ary constraints into unary/binary ones.

• Example: constraint on $X, Y, Z$ with domains:

$$X \in \{1, 2\}, Y \in \{3, 4\}, Z \in \{5, 6\}$$

• Create *encapsulated variable* $U$

Cartesian product $U = X \times Y \times Z$

$$U \in \begin{cases} 
(1, 3, 5), (1, 3, 6), (1, 4, 5), (1, 4, 6), \\
(2, 3, 5), (2, 3, 6), (2, 4, 5), (2, 4, 6) \end{cases}$$
Equivalent binary CSP

• Constraints:
  \[ X + Y = Z \]
  \[ X < Y \]

• Encapsulations
  \[ U \triangleq X \times Y \times Z \]


\[ U[0] = X \]

\[ U[1] = Y \]

\[ X < Y \]
Another example: House puzzle

• A row of 5 houses, each one
  • has a color
  • contains a person with a nationality
  • has a household favorite candy
  • has a household favorite drink
  • contains a pet

• all attributes are distinct

• How should we represent this?
House Puzzle Constraints

• The Englishman lives in the red house.
• The Spaniard owns the dog.
• The Norwegian lives in the first house on the left.
• The green house is immediately to the right of the ivory house.
• The man who eats Hershey bars lives in the house next to the man with the fox.
• Kit Kats are eaten in the yellow house.
• The Norwegian lives next to the blue house.
• The Smarties eater owns snails.
House Puzzle Constraints

• The Snickers eater drinks orange juice.
• The Ukranian drinks tea.
• The Japanese eats Milky Ways.
• Kit Kats are eaten in a house next to where the horse is kept.
• Coffee is drunk in the green house.
• Milk is drunk in the middle house.

Answer the questions:
Where does the zebra live?
Which house drinks water?
House Puzzle Representation

• Variables – What’s common to each thing?

• Domains – What are the domains?
House Puzzle representation

• Constraints are location based, e.g. milk is drunk in the middle house.

• Could we associate variables with a location?

• If so, what are
  • our variables?
  • their domains?
  • and how do we write our constraints?
House puzzle representation

• Colors: red, green, ivory, yellow, & blue
• Nationalities: English, Spaniard, Norwegian, Ukranian, and Japanese
• Pets: dog, fox, snails, horse, and zebra
• Candies: Hershey bars, Kit Kats, Smarties, Snickers, and Milky Way
• Drinks: orange juice, tea, coffee, milk, and water

Note: water and zebra were inferred from the questions
House puzzle representation

Some examples

• Milk is drunk in the middle house.
  \[ \text{milk} = 3 \]

• Coffee is drunk in the green house
  \[ \text{coffee} = \text{green} \]

• Kit Kats are eaten in a house next to where the horse is kept.
  \[ \text{abs(kit kats – horse)} = 1 \]

• The green house is immediately to the right of the ivory home.
  \[ \text{green} = \text{ivory} + 1 \]

• The Norwegian lives next to the blue house
  \[ \text{Norwegian} = \text{blue} + 1 \text{ or } \text{Norwegian} = \text{blue} – 1 \]

• The Norwegian lives in the first house on the left
  \[ \text{Norwegian} = 1 \]
Implementing a CSP problem: Representation

• variables – simple list

• values – Mapping from variables to value lists
e.g. Python dictionary

• neighbors – Mapping from variables to list of other variables that participate in constraints

• binary constraints
  • explicit value pairs
  • functions that return a boolean value
Representation of house problem

• variables:
  list of colors, nationalities, pets, candies, & drinks
  \{red, green, ivory, yellow, blue, English, Spaniard, \ldots\}

• values: \( X_i \in \{1,2,3,4,5\} \)
  except milk = \{3\}, Norwegian = \{1\}

• neighbors:
  • all variable pairs from constraints, e.g. Englishman & red
  • alldiff(red, green, ivory, blue), alldiff(English, Spaniard, \ldots), other category
    alldiffs
Representation of house problem

• constraints – Function $f(A, a, B, b)$
  
  where $A$ and $B$ are variables with values $a$ and $b$ respectively.
  
  Returns true if constraint is satisfied, otherwise false
  
  Example: $f(“Englishman”, 4, “red”, 5)$ returns false as the Englishman lives in the red house.
Let’s think about inference

• We know: Norwegian = {1}, milk = {3}
• Consider:
  Norwegian = blue + 1 or Norwegian = blue − 1

  with a sprinkle of algebra we have:
  blue = Norwegian − 1 or blue = Norwegian+1
  or as Norwegian can only be 1:
  blue = 1-1 or blue = 1+1

• We know blue ∈ {1,2,3,4,5}, therefore blue=2
• Due to the alldiff constraint, we also know:
  \[ \forall color_{color\neq blue color} \in \{1,3,4,5\} \]
  \[ \forall natl_{natl\neq Norwegian natl} \in \{2,3,4,5\} \]
Another inference example

• We have the constraint: green = ivory + 1 and we know that green & ivory in {1,3,4,5}

• Suppose green=3, can the constraint hold?
• What about green=1?
• We can deduce: green ∈ {4,5}
• What about ivory?
How do we formally tame this beastie?

General strategies

• **Local consistency:** Reduce set of possible values through constraint enforcement and propagation
  • node consistency
  • arc consistency
  • path consistency

• Perform search on remaining possible states
Node consistency

• A variable is *node-consistent* if all values satisfy all unary constraints

\[
\text{fruits} = \{ \text{apples, oranges, strawberries, peaches, pineapple, bananas} \}
\]

Condition: \( \text{allergic(TreeBornFruit)} \)
Reduced domain: \( \{\text{strawberries, pineapple}\} \)

• Other unary conditions could further restrict the domain
Arc consistency

- *arc-consistent*
  - variable - all binary constraints are satisfied for the variable
  - network – all variables in CSP are arc-consistent

- Arc consistency only helps when some combinations of values preclude others...
Arc Consistency

Each territory has domain 
\{orange, green, blue\}

WA \neq SA:
\{(orange, green), (orange, blue), (green, orange),
  (green, blue), (blue, orange), (blue, green)\}

Does this reduce the domain of WA or SA?
Arc Consistency

• Constraints that eliminate part of the domain can improve arc consistency

• Variables that represent task starting times

T1 = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
T2 = {2, 3, 4, 5, 6, 7, 8, 9}

• Constraint: $T1 + 5 < T2$ yield consistent domains

T1 = {0, 1, 2, 3, 4}
T2 = {6, 7, 8, 9}
AC-3 arc consistency algorithm

AC3(CSP):

"CSP(variables X, domains D, constraints C)"
q = Queue(binary arcs in CSP)
while not q.empty():
    (Xi, Xj) = q.dequeue()  # get binary constraint
    if revise(CSP, Xi, Xj):
        if domain(Xi) = ∅ return False
        else:
            for each Xk in {neighbors(Xi)- Xj}:
                q.enqueue(Xk, Xi)
    return True

O(cd³) worst case complexity (c # constraints, d max domain size)
AC-3 arc consistency

revise(CSP, X_i, X_j):

“Restrict domain X_i such that it is consistent with X_j”

revised = False
for each x in domain(X_i):
    if not ∃y∈domain(X_j) such that
        constraint holds between x & y:
        delete x from domain(X_i)
    revised = True

return revised

Note: In your text, D_i is used instead of domain(X_i). They are the same thing and both represent the domain values that have not yet been eliminated. This is the current domain of the variable, not the original one.
Path and k-consistency

• Higher levels of consistency, beyond our scope

• General ideas:
  • Path consistency
    See if a pair of variables \( \{X_i, X_j\} \) consistent with a 3rd variable \( X_k \). Solved similarly to arc consistency
  
  • K-consistency
    Given k-1 consistent variables, can we make a k-th variable consistent (generalization of consistency)
Global constraints

Consider the “all different” constraint.

• Each variable has to have a distinct value.

• Assume m variables, and n distinct values.

• What happens when m > n?
Global constraints

Extending this idea:

• Find variables constrained to a single value
• Remove these variables and their values from all variables.
• Repeat until no variable is constrained to a single value
• Constraints cannot be satisfied if
  1. A variable remains with an empty domain
  2. There are more variables than remaining values
Resource constraints ("atmost")

\[
\text{atmost}(20, X, Y, Z) \rightarrow X + Y + Z \leq 20
\]

\[
\text{atmost}(10, P_1, P_2, P_3, P_4) \rightarrow \sum_{i=1}^{4} P_i \leq 10
\]

• Consistency checks
  • Minimum values of domains satisfy constraints?
    • \( P_i = \{3, 4, 5, 6\} \) as \( 3 + 3 + 3 + 3 = 12 \not\leq 10 \)

• Domain restriction
  • Are the largest values consistent with the minimum ones?
    • \( P_i = \{2, 3, 4, 5, 6\} \) as \( 2 + 2 + 2 + (5 \text{ or } 6) = (11 \text{ or } 12) \not\leq 10 \)
**Range bounds**

- Impractical to store large integer sets

- Ranges can be used \([\text{min}, \text{max}]\) instead

- Bounds propagation can be used to restrict domains according to constraints
  - \(X\) domain \([25, 100]\) \quad \([75, 100]\)
  - \(Y\) domain \([50, 125]\) \quad \([100, 125]\)

  How did we get \([75, 100]\)? \(Y = 125 \rightarrow X \geq 75\)
Sudoku

- Puzzle game played with digit symbols
- All-different constraints exist on units
- Some cells initially filled in

- Hard for humans, pretty simple for CSP solvers
Sudoku
Sudoku

Sample constraints

- $\text{Alldiff}(A_1,A_2,A_3,A_4,A_5,A_6,A_7,A_8,A_9)$
- $\text{Alldiff}(A_1,B_1,C_1,D_1,E_1,F_1,G_1,H_1,I_1)$
- $\text{Alldiff}(A_1,A_2,A_3,B_1,B_2,B_3,C_1,C_2,C_3)$

These can be expanded to binary constraints, e.g. $A_1 \neq A_2$
Sudoku

AC-3 constraint propagation

• E6: \( d=\{1, 2, 3, 4, 5, 6, 7, 8, 9\} \)

• Box constraints:
  \( d_1 = d - \{1, 2, 7, 8\} = \{3, 4, 5, 6, 9\} \)

• Column constraints:
  \( d_2 = d_1 - \{2, 3, 5, 6, 8, 9\} = \{4\} \)

Therefore E6=4
Sudoku

AC-3 constraint propagation

- I6: \( d = \{1, 2, 3, 4, 5, 6, 7, 8, 9\} \)
- Column constraints:
  \( d_1 = d - \{2, 3, 4, 5, 6, 8, 9\} = \{1, 7\} \)
- Row constraints:
  \( d_2 = d_1 - \{1, 3, 5\} = \{7\} \)

Therefore I6=7

For this puzzle, continued application of AC-3 would solve the puzzle (not always true)
Naked sets

- Yellow squares form a *naked pair* \{1, 5\}
  - one must contain 1
  - other 5
- Can subtract 1 and 5 from domains of all other cells in row unit.
- These types of “tricks” are not limited to Sudoku puzzles.
Back to searching...

• Once all constraints have been propagated, search for a solution.

• Naïve search
  • Action picks a variable and a value. $n$ variables domain size $d \rightarrow n \cdot d$ possible search nodes
  • Search on next variable.
  • Backtrack when search fails.

• Problems with naïve search
  • $n$ variables with domains of size $d$
  • $nd$ choices for first variable, $(n - 1)d$ for second....

$$nd \cdot (n - 1)d \cdot \ldots \cdot 2d \cdot 1d = n! \cdot d^n$$

leaves but there are only $d^n$ possible assignments!
Back to searching

• CSPs are commutative
• Order of variable selection does not affect correctness (may have other impacts)
• Modified search
  • Each level of search handles a specific variable.
  • Levels have d choices, leaving us with $d^n$ leaves
def backtracking-search(CSP):
    return backtrack({}, CSP);  # call w/ no assignments

def backtrack(assignment, CSP):
    if all variables assigned, return assignment
    var = select-unassigned-variable(CSP, assignment)
    for each value in order-domain-values(var, assignment, csp):
        if value consistent with assignment:
            assignment.add({var = value})
            # propagate new constraints (will work without, but probably slowly)
            inferences = inference(CSP, var, assignment)
            if inferences ≠ failure:
                assignment.add(inferences)
                result = backtrack(assignment, CSP)
                if result ≠ failure, return result
                # either value inconsistent or further exploration failed
                # restore assignment to its state at top of loop and try next value
                assignment.remove({var = value}, inferences)
        # No value was consistent with the constraints
        return failure
Backtracking search

• Several strategies have been employed so far to make searches more efficient, e.g.
  • heuristics (best-first and A* search)
  • pruning (alpha-beta search)

• Can we come up with strategies to improve CSP search?
select-unassigned-variable

• Could try in order: \( \{X_1, X_2, \ldots, X_n\} \)
  Rarely efficient...

• Fail-first strategies
  • Minimum remaining value heuristic:
    Select the most constrained value; the one with the smallest domain.
    Rationale – probably the most likely variable to fail

  • Degree heuristic:
    Use the variable with the highest number of constraints on other unassigned variables.
• Minimum remaining value usually is a better performer than degree heuristic, but not always:

All variables have domains of size three at start, but degree of constraints differs.
order-domain-values

- The order of the values within a domain may or may not make a difference
- Order has no consequence
  - if goal is to produce all solutions or
  - if there are no solutions
- In other cases, we use a fail-last strategy
  - Pick the value that reduces neighbors’ domains as little as possible.

Why fail-first for variable selection and fail-last for value selection?
inference in search

- forward-checking
  - Check arc consistency with neighboring variables.
  - We will see that maintaining arc-consistency (variant of AC3) is more powerful as it propagates all the way through the graph as opposed to forward checking that just looks at neighbors.
forward-checking example

Note: Variable selection is not by degree ordering or min. remaining value
forward-checking example

with minimum remaining value heuristic

<table>
<thead>
<tr>
<th></th>
<th>WA</th>
<th>NT</th>
<th>Q</th>
<th>NSW</th>
<th>V</th>
<th>SA</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial domains</td>
<td>RGB</td>
<td>RGB</td>
<td>RGB</td>
<td>RGB</td>
<td>RGB</td>
<td>RGB</td>
<td>RGB</td>
</tr>
<tr>
<td>After WA=R</td>
<td><strong>R</strong></td>
<td>GB</td>
<td>RGB</td>
<td>RGB</td>
<td>RGB</td>
<td>GB</td>
<td>RGB</td>
</tr>
<tr>
<td>After SA=G</td>
<td>R</td>
<td>B</td>
<td>RB</td>
<td>RB</td>
<td>RB</td>
<td><strong>G</strong></td>
<td>RGB</td>
</tr>
<tr>
<td>After Q=R</td>
<td>R</td>
<td>B</td>
<td><strong>R</strong></td>
<td>B</td>
<td>RB</td>
<td>G</td>
<td>RGB</td>
</tr>
<tr>
<td>After V=R</td>
<td>R</td>
<td>B</td>
<td>R</td>
<td>B</td>
<td><strong>R</strong></td>
<td>G</td>
<td>RGB</td>
</tr>
</tbody>
</table>
forward-checking example

When we assigned SA=G, we restricted NT to B. However, Q was only restricted to R B.

Forward checking does not check anything other than constraints with the neighbor being assigned.

<table>
<thead>
<tr>
<th></th>
<th>WA</th>
<th>NT</th>
<th>Q</th>
<th>NSW</th>
<th>V</th>
<th>SA</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial domains</td>
<td>RGB</td>
<td>RGB</td>
<td>RGB</td>
<td>RGB</td>
<td>RGB</td>
<td>RGB</td>
<td>RGB</td>
</tr>
<tr>
<td>After WA=R</td>
<td>R</td>
<td>GB</td>
<td>RGB</td>
<td>RGB</td>
<td>RGB</td>
<td>GB</td>
<td>RGB</td>
</tr>
<tr>
<td>After SA=G</td>
<td>R</td>
<td>B</td>
<td>RB</td>
<td>RB</td>
<td>RB</td>
<td>G</td>
<td>RGB</td>
</tr>
<tr>
<td>After Q=R</td>
<td>R</td>
<td>B</td>
<td>R</td>
<td>B</td>
<td>RB</td>
<td>G</td>
<td>RGB</td>
</tr>
<tr>
<td>After V=R</td>
<td>R</td>
<td>B</td>
<td>R</td>
<td>B</td>
<td>R</td>
<td>G</td>
<td>RGB</td>
</tr>
</tbody>
</table>
Maintaining arc consistency (MAC)

- Algorithm that propagates constraints beyond the node.
- AC3 algorithm with modified initial queue
  - typical AC3 – *all constraints*
  - MAC – constraints between selected variable $X_i$ and its neighbors $X_j$
    $$\{(X_j, X_i) : X_j \text{ neighbor}(X_i)\}$$
Intelligent backtracking

• Suppose variable ordering: Q, NSW, V, T, SA, WA, NT
• and assignments: 
  \{Q=red, NSW=green, V=blue, T=red\}
• SA is problematic...
  • backtracking will try new values for Tasmania

• What if we could backjump to the variable that caused the problem?
Conflict-directed backjumping

• Maintain a *conflict set* for each variable $X$: A set of assignments that restricted values in $X$’s domain.

• When a conflict occurs, we backtrack to the last conflict that was added.

• In the case of SA,
  • assignments to $Q$, $NSW$, and $V$ restricted SA’s domain
  • variable ordering: $Q$, $NSW$, $V$, $T$, $SA$, $WA$, $NT$
  • SA conflict set \{Q=red, NSW=green, V=blue\}
  • so we backjump to SA’s last conflict $V$. with \{Q=red, NSW=green\}
Backjumping implementation

• On forward checks of X assigned to x,
  • when X deletes a value from Y’s domain, add X=x to Y’s conflict set
  • If Y is emptied, add Y’s conflict set to X’s and backjump to the last added conflict.
    Adding these lets us be smarter about where to backjump.

• Easy to implement, build conflict set during forward check.
Backjumping

• What we prune in conflict-directed backjumping is redundant to what we’d prune from forward checking or MAC searches.
• Interesting, but better to just use forward checking/MAC...

• Still a good idea, what if we could extend it?
More sophisticated backjumps...

• Assignments to the right are inconsistent
  • Suppose we try and assign T, NT, Q, V, SA
  • SA, NT, Q have reduced domains
    \{green, blue\} and cannot be assigned
  • Backjumping fails when a domain is reduced to \(\emptyset\) as SA, NT, and Q are consistent with WA, NSW.

• Can we determine that there is a *conflict set* \{WA, SA, NT, Q\} that are causing the issue?
Conflict-directed backjumps

- Variable order: WA, NSW, T, NT, Q, V, SA
- SA fails. \( \text{conf}(SA) = \{\text{WA}=\text{red}, \text{NT}=\text{blue}, \text{Q}=\text{green}\} \)
- Last variable in \( \text{conf}(SA) \) is Queensland
  - Absorb SA’s conflict set into Q
    \[
    \text{conf}(Q) = \text{conf}(Q) \cup \text{conf}(SA) - \{Q\}
    \]
  - \( \text{conf}(Q) \)
    \[
    = \{\text{NT}=\text{blue}, \text{NSW} = \text{red}\} \cup \{\text{WA}=\text{red}, \text{NT}=\text{blue}, Q=\text{green}\} - \{Q=\text{green}\}
    
    = \{\text{WA}=\text{red}, \text{NSW}=\text{red}, \text{NT}=\text{blue}\}
    
    \]
    Unable to assign a different color to Q, backjump
  - \( \text{conf}(NT) = \text{conf}(NT) \cup \text{conf}(Q) - \{\text{NT}\} \)
    \[
    = \{\text{WA}=\text{red}\} \cup \{\text{WA}=\text{red}, \text{NSW}=\text{red}, \text{NT}=\text{blue}\}
    
    = \{\text{WA}=\text{red}, \text{NSW}=\text{red}\}
    
    \]
    When we run out of colors for NT, we will backjump to NSW

Note: \( \text{conf}(SA) \) would have had NSW=red if NSW was processed before WA
Constraint-learning and no-goods

• On the Australia CSP, we identified a minimal set of assignments that caused the problem.

• We call these assignment *no-goods*.

• We can avoid running into this problem again by adding a new constraint (or checking a no-good cache).
Local Search CSPs

• Alternative to what we have seen so far
• Assign everything at once
• Search changes one variable at a time
  • Which variable?
Min-Conflicts Local Search

def minconflicts(csp, maxsteps):
    current = assign all variables
    for i = 1 to maxsteps:
        if solution(current), return current
        var = select conflicted variable at random from current
        val = find value that minimizes the number of conflicts
        update current such that var=val
    return failure
Min-Conflicts local search

• Pretty effective for many problems, e.g. million queens problem can be solved in about 50 steps

• This is essentially a greedy search, consequently:
  • local extrema
  • can plateau
  • many techniques discussed for hill climbing can be applied (e.g. simulated annealing, plateau search)