Department of
Mathematics &
Computer Science

6. Constraint Satisfaction Problems

Overview

Some problem examples:

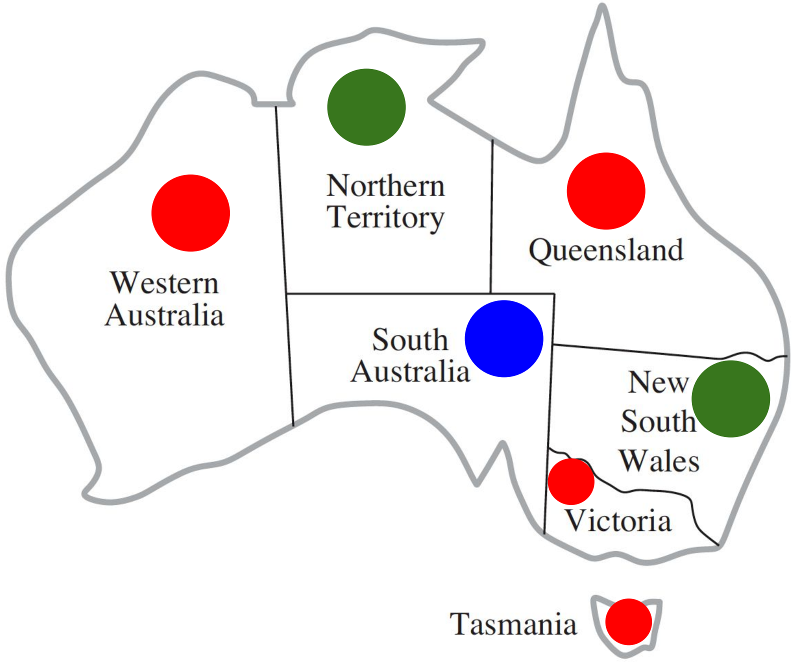
- Map coloring problem
- Car assembly scheduling problem
- Sudoku puzzle

CSP algorithms to solve these problems:

- Arc consistency
- Tree CSP-search algorithm

6.1.1 Australia map coloring problem

Problem: Color each region either red, green, or blue in such a way that no neighboring regions have the same color.



How can we structure this problem so we can apply CSP algorithms?

Example Solution - `{WA=red, NT=green, Q=red, NSW=green, V=red, SA=blue, T =red}`

6.1.1 Formulating map coloring problem as a CSP

Variables are the regions:

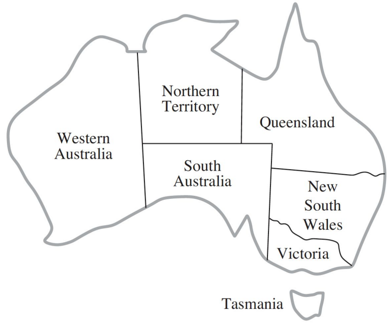
`X = {WA, NT, Q, NSW, V, SA, T}`

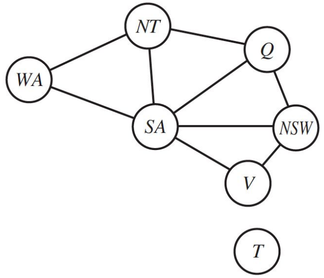
The domain of each variable is the following set:

D_i = {red, green, blue}

The nine constraints (for each regions border):

$C = \{SA \neq WA, SA \neq NT, SA \neq Q, SA \neq NSW, SA \neq V, WA \neq NT, NT \neq Q, Q \neq NSW, NSW \neq V\}$





$SA \neq WA$ is a shortcut for ⟨(SA,WA), {(red , green), (red , blue), (green, red), (green, blue), (blue, red), (blue, green)} ⟩

For example, (SA, WA) can take the values such as (red, green)

{ } - set
( ) - tuple / ordered sequence
⟨⟩- vector

State space approach searching vs CSP

State Space Approach:

- Problems can be solved by searching in a space of states
- These states can be evaluated by domain-specific heuristics and tested to see whether they are goal states
- From the point of view of the search algorithm, however, each state is atomic, or indivisible—a black box with no internal structure

Constraint Satisfaction Approach:

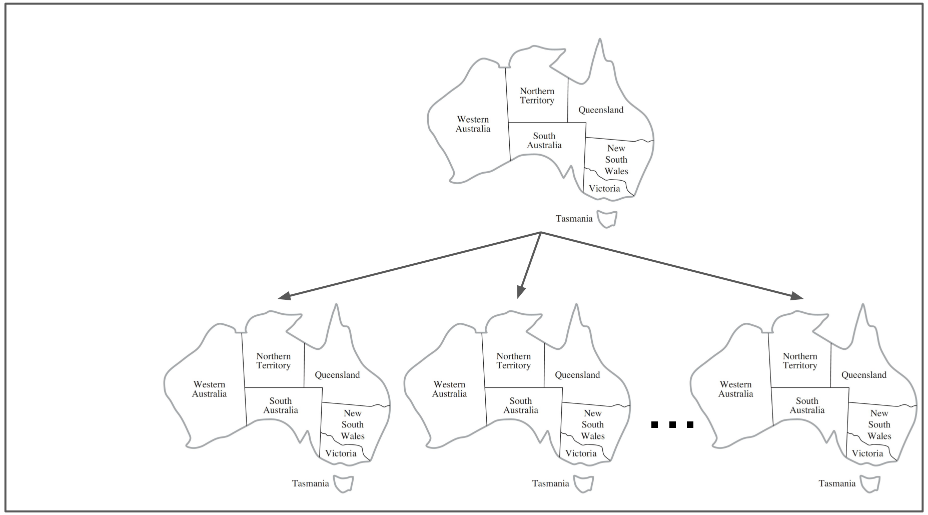
(more efficient for many problems)

- Use a factored representation for each state: a set of variables, each of which has a value
- Problem is solved when each variable has a value that satisfies all the constraints on the variable

CSP algorithms take advantage of the structure of states and use general-purpose heuristics to enable the solution of complex problems.

Example: Shortest Path (global perspective is important) vs Sudoku (local solutions lead to global)

State space search approach



6.1 Defining CSPs

A constraint satisfaction problem consists of three components - `X`, `D`, and `C`:

`X` is a set of variables, $\{X_1, \ldots, X_n\}$

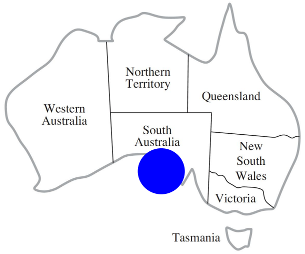
`D` is a set of domains, $\{D_1, \ldots, D_n\}$, one for each variable

`C` is a set of constraints that specify allowable combinations of values

- Each domain D_i consists of a set of allowable values, $\{v_1, \ldots, v_k\}$ for variable X_i
- Each constraint C_i is a pair ⟨`scope, rel`⟩
  - scope is a tuple of variables that participate in the constraint
  - rel is a relation that defines the values that those variables can take on

  Example: $SA \neq WA$ is a shortcut for ⟨`(SA,WA), {(red , green), (red , blue), (green, red), (green, blue), (blue, red), (blue, green)}`⟩
- Each state in a CSP is defined by an assignment of values to some or all of the variables, $\{X_i=v_i, X_j=v_j, \ldots\}$

Why formulate a problem as a CSP?



1. CSPs yield a natural representation for a wide variety of problems
   - If you already have a CSP-solving system, it is often easier to solve a problem using it
     - than to design a custom solution using another search technique
2. CSP solvers can be faster than state-space searchers
   - Because CSP solvers can quickly eliminate large swatches of the search space

Example:

- Once we have chosen `{SA=blue}` in the Australia problem, we can conclude that none of the five neighboring variables can take on the value blue.
- Without taking advantage of constraint propagation, a search procedure would have to consider $3^5 = 243$ assignments for the five neighboring variables
- With constraint propagation we never have to consider blue as a value, so we have only $2^5 = 32$ assignments
- a reduction of 87%

THINK PAIR SHARE

6.1.2 Scheduling the assembly of a car

(Sub-)problem: Scheduling consists of 15 tasks

- Install axles (front and back), affix all four wheels (right and left, front and back), tighten nuts for each wheel, affix hubcaps, and inspect the final assembly.

Variables

- Each task can be modeled as a variable, e.g. install wheel
- Value of the variable = **time that the task starts**
- Example, $\texttt{Axle}_F = 0$ **indicates front axle installation begins at time 0**
- All 15 variables, $X = \{\texttt{Axle}_F, \texttt{Axle}_B, \texttt{Wheel}_{RF}, \texttt{Wheel}_{LF}, \texttt{Wheel}_{RB}, \texttt{Wheel}_{LB}, \texttt{Nuts}_{RF}, \texttt{Nuts}_{LF}, \texttt{Nuts}_{RB}, \texttt{Nuts}_{LB}, \texttt{Cap}_{RF}, \texttt{Cap}_{LF}, \texttt{Cap}_{RB}, \texttt{Cap}_{LB}, \texttt{Inspect}\}$

Constraints

- Assert that one task must occur before another
  - for example, **a wheel must be installed before the hubcap is put on**
- Assert that only so many tasks can go on at once (for example, when sharing tools)
- Specify that a task takes a certain **amount of time to complete**

6.1.2 Constraints for the car assembly problem

It takes 10 minutes to install an axle:

$Axle_F + 10 \leq Wheel_{RF}$; $Axle_F + 10 \leq Wheel_{LF}$; $Axle_B + 10 \leq Wheel_{RB}$; $Axle_B + 10 \leq Wheel_{LB}$

It takes 1 minute to affix the wheels:

$Wheel_{RF} + 1 \leq Nuts_{RF}$; $Wheel_{LF} + 1 \leq Nuts_{LF}$; $Wheel_{RB} + 1 \leq Nuts_{RB}$; $Wheel_{LB} + 1 \leq Nuts_{LB}$

It takes 2 minutes to tighten nuts (before attaching the hubcaps):

$Nuts_{RF} + 2 \leq Cap_{RF}$; $Nuts_{LF} + 2 \leq Cap_{LF}$; $Nuts_{RB} + 2 \leq Cap_{RB}$; $Nuts_{LB} + 2 \leq Cap_{LB}$

Have to share one tool that helps put the axle in place:

We need a **disjunctive constraint** to say that $Axle_F$ and $Axle_B$ must not overlap in time

$(Axle_F + 10 \leq Axle_B)$ or $(Axle_B + 10 \leq Axle_F)$

Inspection takes 3 minutes:

$X + 3 \leq Inspect$ (for each X)

What will be the constraint/s for - "Get the whole assembly done in 30 min"?

THINK PAIR SHARE

Hint: Can this constraint be used to define domains?

Approaches for solving CSPs (Topics we will cover)

- Constraint Propagation / Inference
  - Requires the concepts of node consistency, arc consistency, path consistency, k-consistency
  - Arc consistency can solve an ‘easy’ sudoku problem
- Backtracking search
  - Variable and value ordering
  - Intelligent backtracking
- Local search
- Using the structure of the problem to find quick solutions
  - The Tree-CSP-Search algorithm
  - How to reduce constraint graphs to tree?
    - The cutset conditioning algorithm
    - Tree decomposition approach

6.2.2 What is arc consistency?

Arc consistency is reducing the domain of the variables (X) to consistent values.

Example 1 (works):

Given:

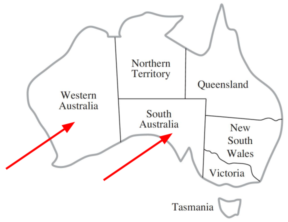
- X and Y are both 'set of digits'
- Constraint: `X = Y`^{2} OR ⟨`(X,Y), {(0,0), (1,1), (2,4), (3,9)}` ⟩

Apply arc consistency:

- To make X arc-consistent with respect to Y
  - we reduce X's domain to `{0, 1, 2, 3}`
- To make Y arc-consistent with respect to X
  - we reduce Y's domain to `{0, 1, 4, 9}`

X_i is arc-consistent with respect to X_j (X with Y) if for every value in the current domain D_i there is some value in domain D_j that satisfies the binary constraint on the arc (X_i, X_j)

6.2.2 What is arc consistency?



Example 2 (does nothing):

Given:

- SA and WA are in the set {red, green, blue}
- The inequality constraint on (SA, WA) is `{(red, green), (red, blue), (green, red), (green, blue), (blue, red), (blue, green)}`

Apply arc consistency:

- To make SA arc-consistent with respect to WA
  - we will reduce SA's domain to `{red, green, blue}`

- A network is arc-consistent if every variable is arc-consistent with every other variable

6.2.6 Sudoku puzzle

The Sudoku Puzzle

- The Sudoku board consists of 81 squares
- Some of the squares are already filled (1 to 9)

Goal

- Fill in the remaining squares such that no digit appears twice in any row, column, or 3x3 box
- There is exactly one solution
- Even the hardest Sudoku problems yield to a CSP solver in less than 0.1 second

	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

	1	2	3	4	5	6	7	8	9
A	4	8	3	9	2	1	6	5	7
B	9	6	7	3	4	5	8	2	1
C	2	5	1	8	7	6	4	9	3
D	5	4	8	1	3	2	9	7	6
E	7	2	9	5	6	4	1	3	8
F	1	3	6	7	9	8	2	4	5
G	3	7	2	6	8	9	5	1	4
H	8	1	4	2	5	3	7	6	9
I	6	9	5	4	1	7	3	8	2

6.2.6 Solving sudoku: Variables, domains, & constraints

- Variables:
  - A1 through A9 for the top row
  - I1 through I9 for the bottom row
- Variable Domains:
  - The empty squares have the domain `{1, 2, 3, 4, 5, 6, 7, 8, 9}`
  - The prefilled squares have a domain consisting of a single value
- Constraints:
  - 27 “Alldiff” constraints: one for each **row**, **column**, and **box of 9 squares**

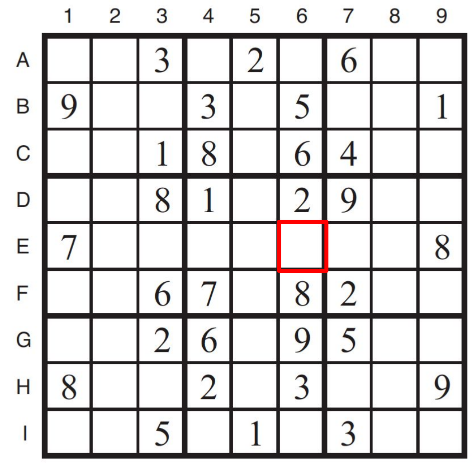
$$
\begin{aligned}
&\mathit{Alldiff}(A1, A2, A3, A4, A5, A6, A7, A8, A9)\\
&\mathit{Alldiff}(B1, B2, B3, B4, B5, B6, B7, B8, B9)\\
&\dots\\
&\mathit{Alldiff}(A1, B1, C1, D1, E1, F1, G1, H1, I1)\\
&\mathit{Alldiff}(A2, B2, C2, D2, E2, F2, G2, H2, I2)\\
&\dots\\
&\mathit{Alldiff}(A1, A2, A3, B1, B2, B3, C1, C2, C3)\\
&\mathit{Alldiff}(A4, A5, A6, B4, B5, B6, C4, C5, C6)\\
&\dots
\end{aligned}
$$

	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

6.2.6 Arc consistency for solving sudoku

Consider the variable E6:

- Initially the domain of E6 is `{1, 2, …, 9}`
- To make E6 arc-consistent with respect to the eight other variables in the box (D4 to F6) we will reduce the domain of E6 to `{3, 4, 5, 6, 9}`
  - i.e. we remove 1, 2, 7, and 8
- Next, to make E6 arc-consistent will the eight variables in the row E, we will reduce the domain of E6
  - In this case, the domain remains same
- Further, to make E6 arc-consistent with eight variables in the column 6, we will reduce the domain of E6 down to `{4}`
  - because 3, 5, 6, and 9 are all in the column
- This solves E6



	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

What is arc consistency? It is reducing the domain of the variables (X) to consistent values (by satisfying constraints).

6.2.6 Arc consistency for solving sudoku

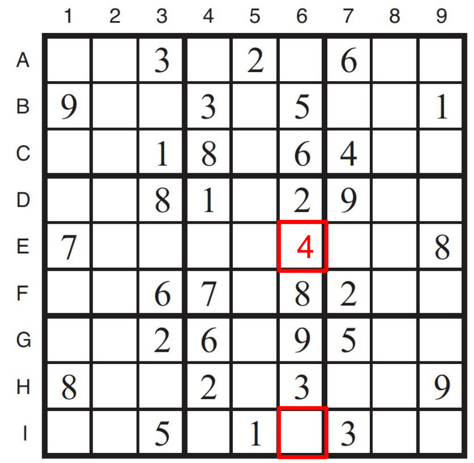
Consider the variable I6:

- I6 = `{1, 2, 3, …, 9}`
- Applying arc consistency in its column, we eliminate 5, 6, 2, 4, 8, 9, and 3
  - I6 = `{1, 7}`
- We eliminate 1 by arc consistency with I5
  - I6 = `{7}`

Consider the variable A6:

- A6 = `{1, 2, 3, …, 9}`
- Applying arc consistency in its column, we eliminate 5, 6, 2, 4, 8, 9, 3, and 7
  - A6 = `{1}`

AC algorithms work only for easiest Sudoku puzzles!



	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7					4			8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

How do 'we' know that we have to solve I6 after E6?

THINK PAIR SHARE

Can Arc-consistency solve the car assembly problem? How?

THINK PAIR SHARE

It takes 10 minutes to install an axle:

$Axle_F + 10 \leq Wheel_{RF}$; $Axle_F + 10 \leq Wheel_{LF}$; $Axle_B + 10 \leq Wheel_{RB}$; $Axle_B + 10 \leq Wheel_{LB}$

It takes 1 minute to affix the wheels:

$Wheel_{RF} + 1 \leq Nuts_{RF}$; $Wheel_{LF} + 1 \leq Nuts_{LF}$; $Wheel_{RB} + 1 \leq Nuts_{RB}$; $Wheel_{LB} + 1 \leq Nuts_{LB}$

It takes 2 minutes to tighten nuts (before attaching the hubcaps):

$Nuts_{RF} + 2 \leq Cap_{RF}$; $Nuts_{LF} + 2 \leq Cap_{LF}$; $Nuts_{RB} + 2 \leq Cap_{RB}$; $Nuts_{LB} + 2 \leq Cap_{LB}$

Have to share one tool that helps put the axle in place:

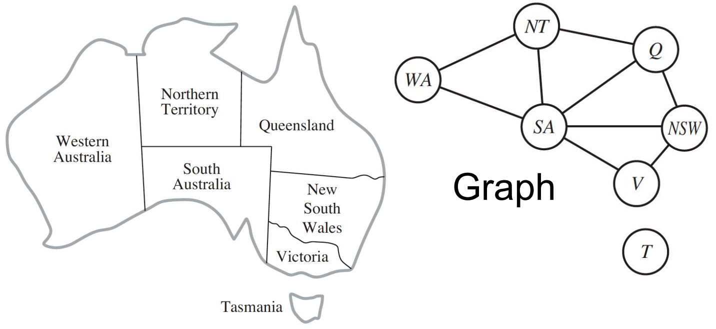
We need a **disjunctive constraint** to say that $Axle_F$ and $Axle_B$ must not overlap in time

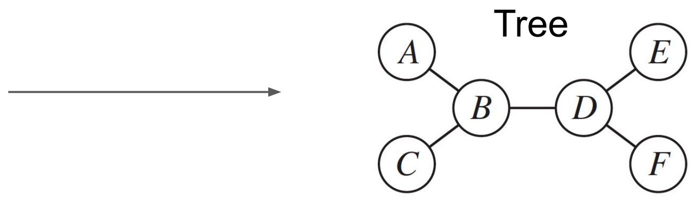
$(Axle_F + 10 \leq Axle_B)$ or $(Axle_B + 10 \leq Axle_F)$

Inspection takes 3 minutes:

$X + 3 \leq Inspect$ (for each X)

6.5 A reduced Australia map coloring problem



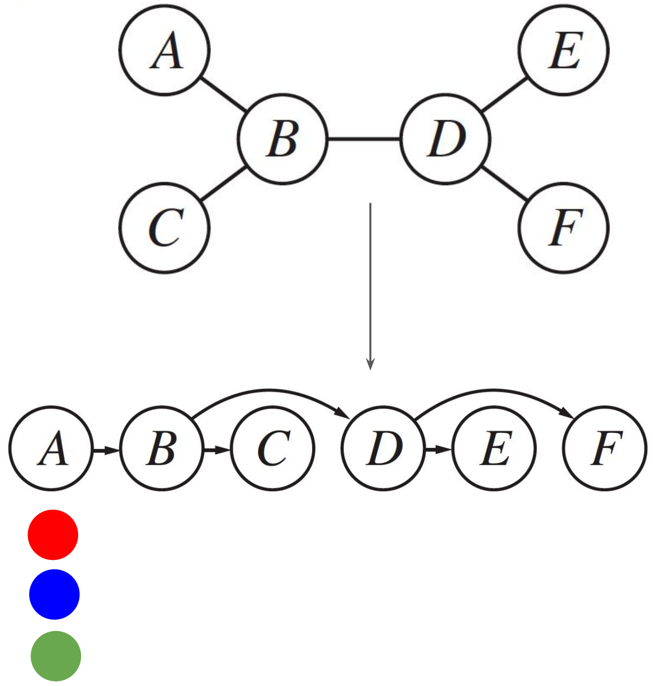


- We would like to color the states A, B, C, D, E, and F with {green, blue, red} colors
- A constraint graph is a tree when any two variables are connected by only one path (no cycles)

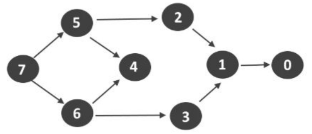
6.5 The TREE-CSP-SOLVER algorithm

```
function TREE-CSP-SOLVER*( csp) returns a solution, or failure
    inputs: csp, a CSP with components $X, D, C$

    $n \leftarrow$ number of variables in $X$
    $assignment \leftarrow$ an empty assignment
    $root \leftarrow$ any variable in $X$
    $X \leftarrow$ TOPOLOGICALSORT($X, root$)
    for $i$ = 1 to $n$ do
        $assignment[X_i] \leftarrow$ any consistent value from $D_i$
        if there is no consistent value then return failure
    return assignment
```



Topological Ordering: Ordering of vertices such that for every directed edge uv from u to v, u comes before v in the ordering



*This is a slightly reduced version of the algorithm in the book

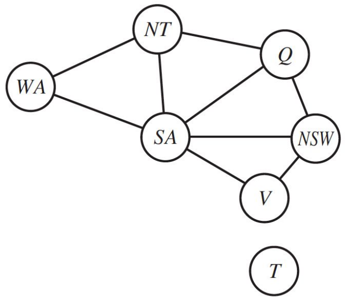
6.5 Reducing graphs to trees

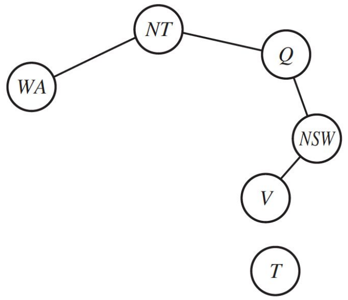
How to reduce a graph to a tree?

- Assign values to some variables, so that the remaining graph becomes a tree

Example:

- If we delete “South Australia” the remaining constraint graph becomes a tree
  - We can SA to “blue” and remove “blue” from the domain of all adjacent states (equivalent to deleting)
  - {SA} is **cycle cutset** because after removing {SA} the constraint graph becomes a tree





Tree is special form of graph i.e. minimally connected graph and having only one path between any two vertices.

- Cycle cutset is a subset of variables in a constraint graph such that after removing the cycle cutset, the graph becomes a tree

6.5 Cutset conditioning algorithm

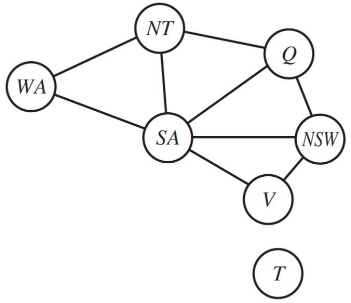
Step 1: Choose a cycle cutset

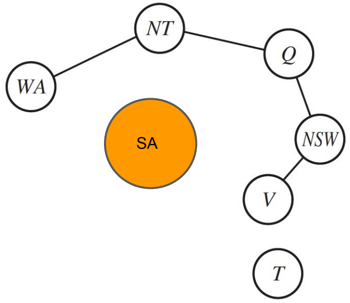
A subset S of the CSP's variables such that the constraint graph becomes a tree after removal of S

Step 2: First solve the cycle cutset then solve the remaining tree

For each possible assignment to the variables in S that satisfies all constraints on S:

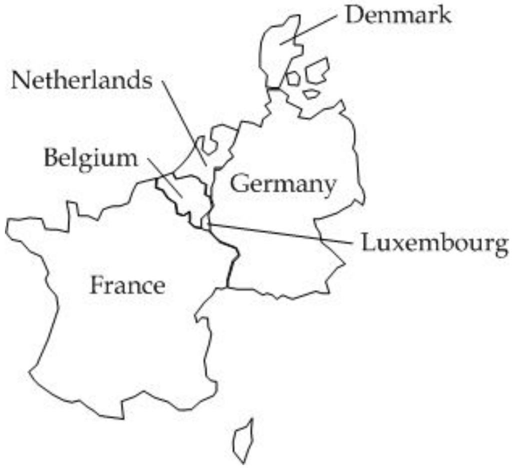
a. Remove from the domains of the remaining variables any values that are inconsistent with the assignment for S, and
b. If the remaining CSP has a solution, return it together with the assignment for S.





Cutset conditioning algorithm example

Sample Problem: Provide the steps (with numbering) how you will apply the Cutset Conditioning Algorithm and the Tree-CSP-Solver algorithm to color the following map with 4 colors - blue, white, red or green - such that no two neighboring countries have the same color. Following the steps, color the map with the 4 colors - blue, white, red or green - such that no two neighbouring countries have the same color.



Steps

1. Draw the constraint graph
2. Apply Cutset Conditioning algorithm
    a. Choose a cycle cutset S (remaining variables for a tree T)
    b. (Loop) Assign constraint 'satisfying' values to S
        i. Update the domain of the variables in T (by deleting all values of the variables in S)
        ii. Apply Tree-CSP-Solver to solve T
            1. Choose root
            2. Make graph directed (assume that we apply BFS)
            3. Build a topological order
            4. Assign colors
        iii. Return solution if T can be solved, i.e. all variables have consistent values

Summary

- Arc consistency algorithm can solve problems such as ‘easy’ Sudoku.
- Cutset conditioning can reduce a general CSP to a tree-structured one and is quite efficient if a small cutset can be found.