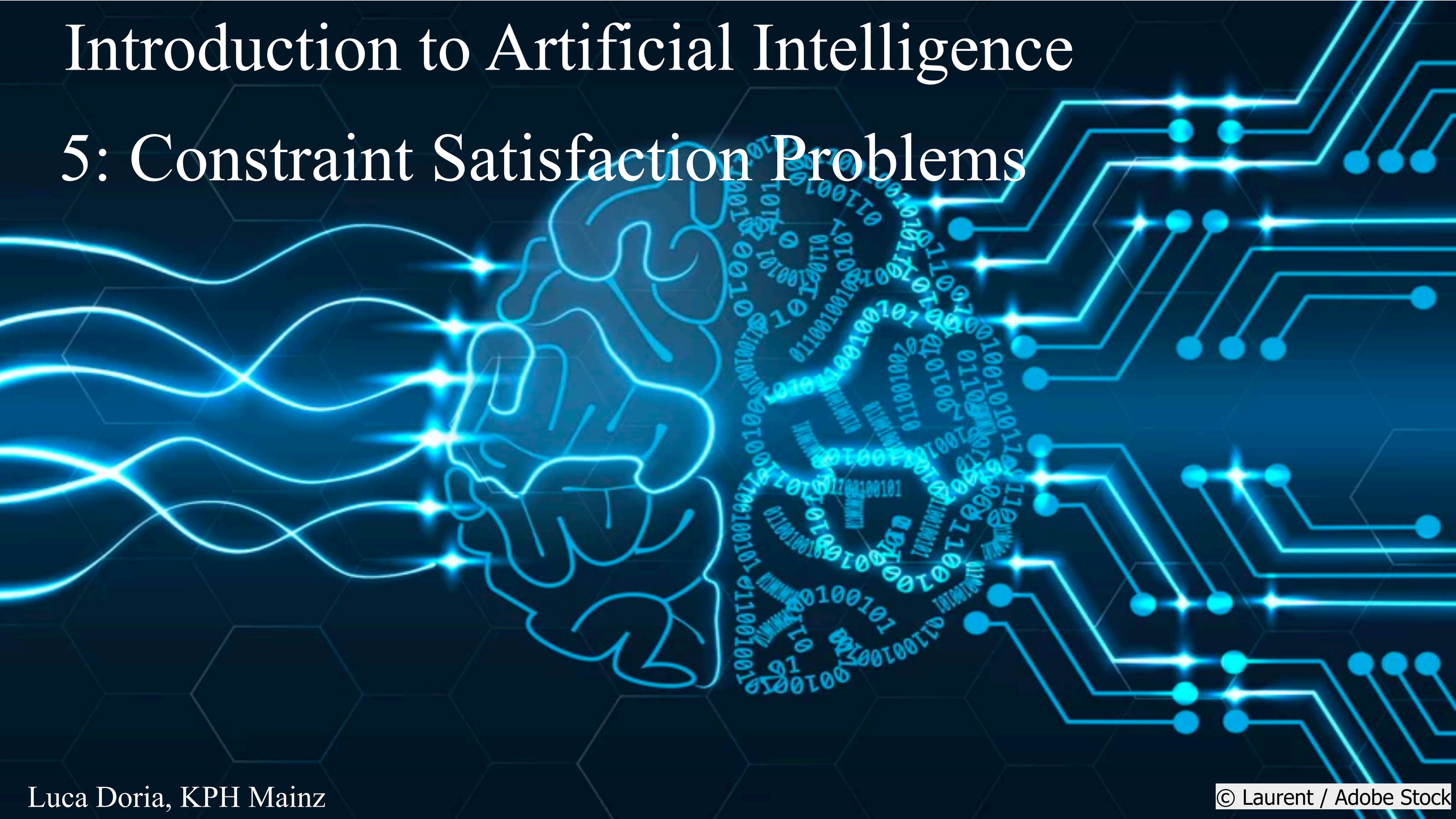
Introduction to Artificial Intelligence

5: Constraint Satisfaction Problems



Luca Doria, KPH Mainz

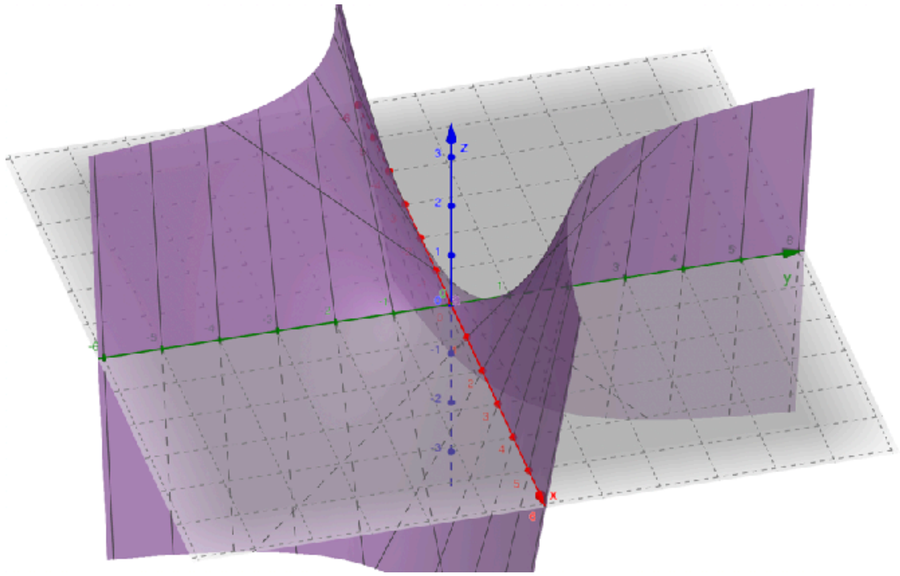
Problems with Constraints

Constraint satisfaction problems (CSP) often deal with the optimisation of a function together the satisfaction of constraints.
In the continuum, it is a classical mathematical problem like for example:

Find the maximum of f(x,y) = xy , given the constraint x+y=10.

This translates to minimising the "lagrangian" $\mathcal{L} = xy + \lambda(x + y - 10)$

$$\begin{cases} \dfrac{\partial \mathcal{L}}{\partial x} = y + \lambda = 0 \\ \dfrac{\partial \mathcal{L}}{\partial y} = x + \lambda = 0 \\ \dfrac{\partial \mathcal{L}}{\partial \lambda} = x + y - 10 = 0 \end{cases} \Rightarrow \quad x = y = 5$$



Constraint Satisfaction Problems

- A Constraint Satisfaction Problems (CSP) is given by a set of variables x_1, x_2, ..., x_n, an associated set of value domains dom_1, dom_2, ..., dom_n, and a set of constraints. i.e., relations, over the variables.

- An assignment of values to variables that satisfies all constraints is a solution of such a CSP.

- In CSPs viewed as search problems, states are explicitly represented as variable assignments. CSP search algorithms take advantage of this structure.

- The main idea is to exploit the constraints to eliminate large portions of search space.

- Formal representation language with associated general inference algorithms

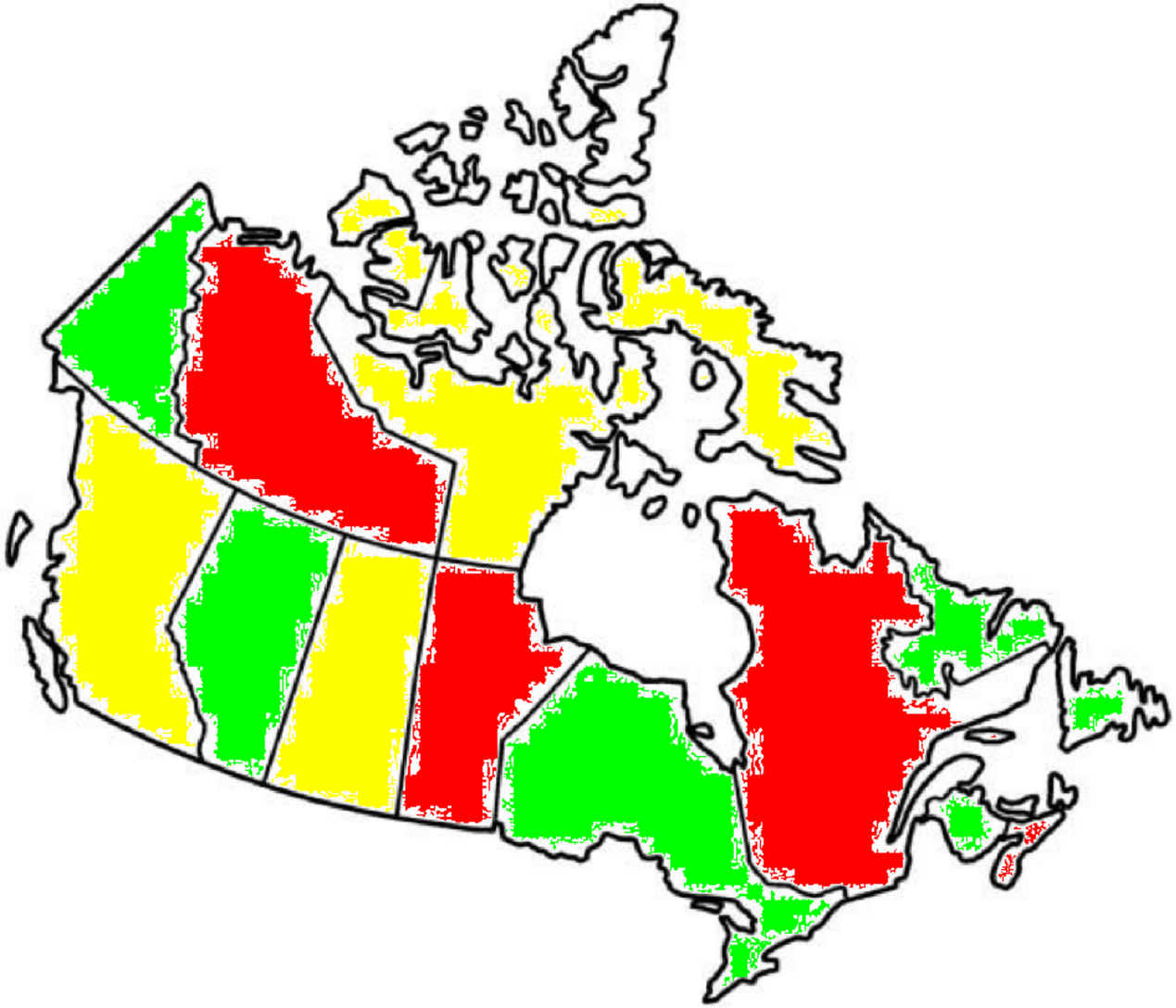
Example: Map Coloring

Variables: NL, PE, NS, NB, QC, ON, MB, SK, AB, BC, YT, NY, NU

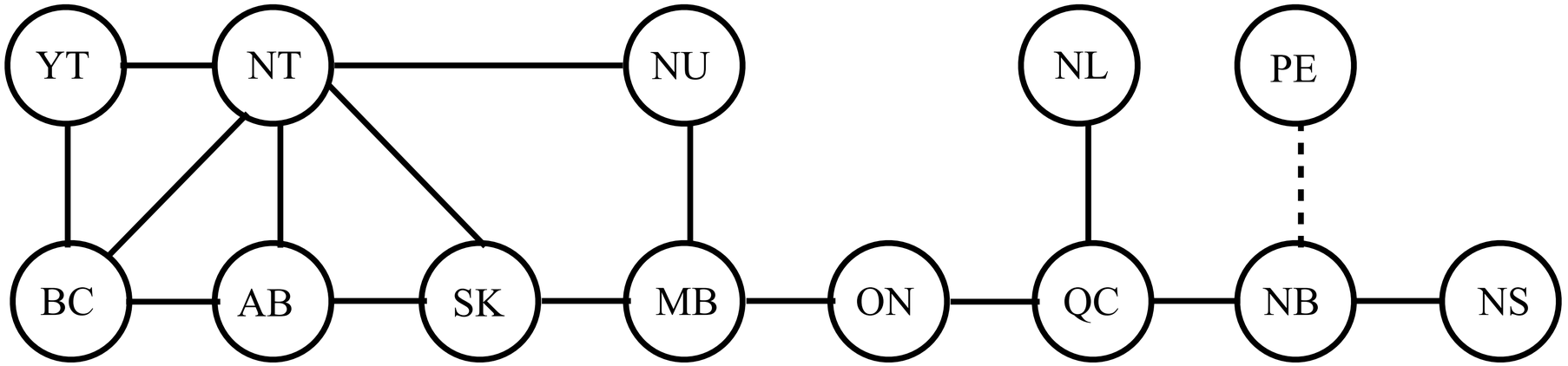
Values: Red, Green, Blue

Constraints: adjacent regions must have different colors.

Possible Solution



Constraint Graphs



- A constraint graph can be used to visualise binary constraints
- For high-order constraints, hyper graph representations are available
- Nodes = variables
- Arcs = constraints

Other variations

- Binary, ternary, or even higher arity (e.g., ALL DIFFERENT)
- Finite domains (d values) → d^n possible variable assignments
- Infinite domains (reals, integers)
- linear constraints (each variable occurs only in linear form): solvable (in P if real)
- nonlinear constraints: unsolvable

Applications

- Timetabling (classes, rooms, times)
- Configuration (hardware, cars, . . .)
- Scheduling
- Floor planning
- Frequency assignments
- Sudoku

...

Constraint Propagation

A CSP algorithm can be seen as listing a graph where variables are nodes and constraints are edges.

In expanding a node, the algorithm can:

- assign a variable (e.g. decide the colour in the map example)
- do constraint propagation: use the constraints to reduce the choices for the variable

Many possibilities:

- Local consistency: general idea of checking the constraints connected to the current node
- Node consistency: check and resolve all the unary constraits (e.g: BC people do not like red)
- Arc consistency: check and resolve all the binary constraints (e.g.: no same neighbouring colours)
- Path consistency: similar to AC, but over 3 nodes
- k-consistency: for a consistent assignment of k-1 variables, there is a constant assignment of the k-th variable.

Additional note: remember that CSPs can have also global constraints

AC-3: An Arc-Consistency Algorithm

Developed by A. Mackworth (1977), the name comes from being the 3rd version developed in the original paper.

AC-3 tries to make all variables arc-consistent using a queue of arcs to consider.

Initially, the queue contains all the arcs, where binary constrains become 2 arcs (1 for each direction)

AC-3 pops an arc from the queue (x_i,x_j) and makes xi arc-consistent.

If D_i (the domain of x_i) does not change, move to the next arc.

If D_i changes, add to the queue all the arcs (x_k,x_i) where x_k is a neighbour of x_i.

—> Why? Because the reduction of D_i might imply reductions in D_k later on.

If Di becomes empty, return FAIL.

Otherwise, keep inspecting the queue until empty.

Worst-case time complexity O(#constraints * domain-size^3)

AC-3: An Arc-Consistency Algorithm

```
function AC3 (csp) : returns false if inconsistency is found, otherwise true
  queue <— arcs (initially all the arcs of csp)
      while queue not empty do
            (xi,xj) <— POP(queue)
            if REVISE(csp,xi,xj) then
                  if size(Di=0) then return false
                  for each xk in xi.NEIGHBORS - {xj} do
                        add (xk,xi) to queue
       return true
```

```
function REVISE(csp,xi,xj): returns true iff the domain of xi is revised
      revised <— false
      for each x in Di do
            if no value y in Dj allows (x,y) to satisfy the constraint between xi and xj then
                  delete x from Di
                  revised <— true
        return revised
```

Another example: SUDOKU

81 variables

$D_i = \{1,2,3,4,5,6,7,8,9\}$

9+9+9=27 Alldiff constraints:

All the rows must be different (9):

Alldiff(A1,A2,A3,A4,A5,A6,A7,A8,A8)

Alldiff(B1,B2, ….)

….

All columns must be different (9)

All squares must contain different numbers (9)

AC-3 can solve the easiest puzzles, while for the more complex ones one has to apply longer path-consistency algorithms.

6	1	3		2		5	4	
2				4	6		3	1
				3		6	7	2
	6	1						
	7			6	9		1	5
4		9	5	8			6	
			8	1	2	3		
	9	6				2	8	
	3						5	7

After consistency: search!

After the constraint propagation process, we might still not have a solution and we need to search for one. Considering the search tree:

- For a CSP with n variables with domain size d and the total possible assignments are d^n.
- The solution must be at depth n.
- The branching factor at the root node is nd.
- At the deeper level, the branching factor is (n-1)d and so on..
- This means that in total the number of tree leaves is $n!d^n$.

Remember that initially we expected d^n assignment but we found $n!d^n$.

The factor n! comes from the commutativity property of the CSP.

Commutativity refers to the fact that we can commute the variable's values and obtain still a valid solution. Example: in the map colouring we can start with whatever of the available colours.

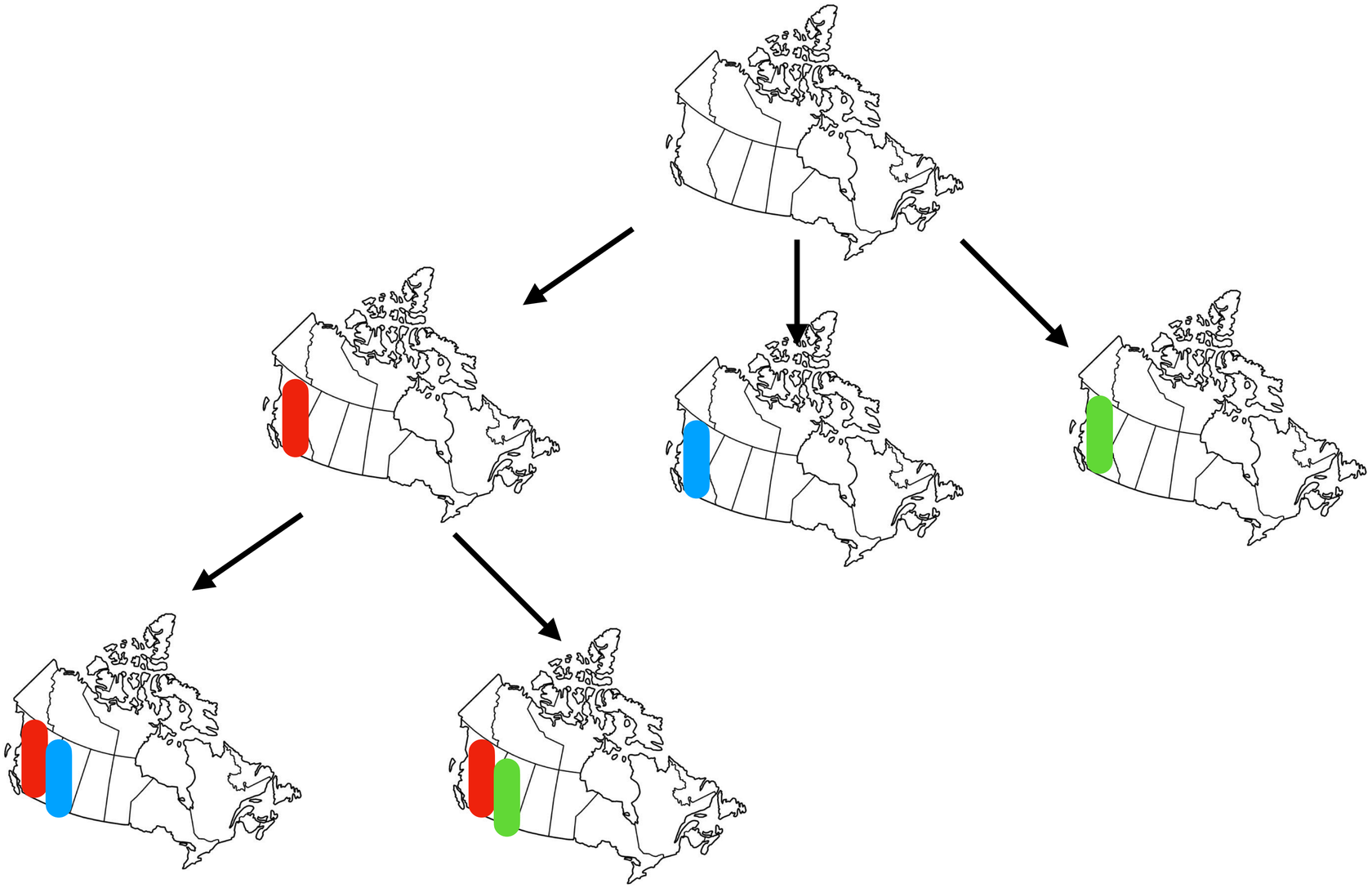
Backtracking (recursive depth-first search)

```
function BACKTRACKING-SEARCH(csp) returns a solution, or failure
    return BACKTRACK(csp,{})
```

```
function BACKTRACK(csp,assignment) returns a solution or failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(csp,assignment)
    for each value in ORDER-DOMAIN-VALUES(csp,var,assignment) do
        if value is consistent with assignment then
            add {var = value} to assignment
            inferences ← INFERENCE(csp , var , assignment )
            if inferences != failure then
                add inferences to csp
                result←BACKTRACK(csp, assignment)
                if result != failure then return result
                remove inferences from csp
            remove {var = value} from assignment
return failure
```

Optional consistency check, e.g. arc-consistency

Map-coloring Example



Efficiency Improvement

- Variable ordering: Which one to assign first?
- Value ordering: Which value to try first?
- Try to detect failures early on
- Try to exploit problem structure

All this is not problem-specific…

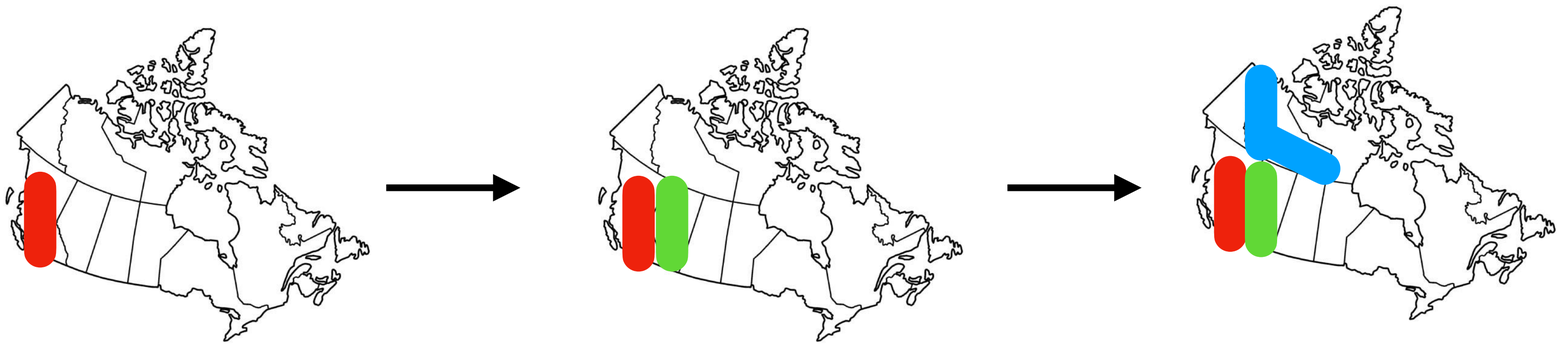
Variable Ordering

In the BACKTRACKING algorithm we have to choose a variable with SELECT-UNASSIGNED-VARIABLE.

Simple solutions (which are not optimal!)

- Order assignment
- Random assignment

Better strategy: Minimum-remaining-values heuristic (MRV). Idea: choose the variable leading to fewest allowed moves. This reduces the branching fraction.



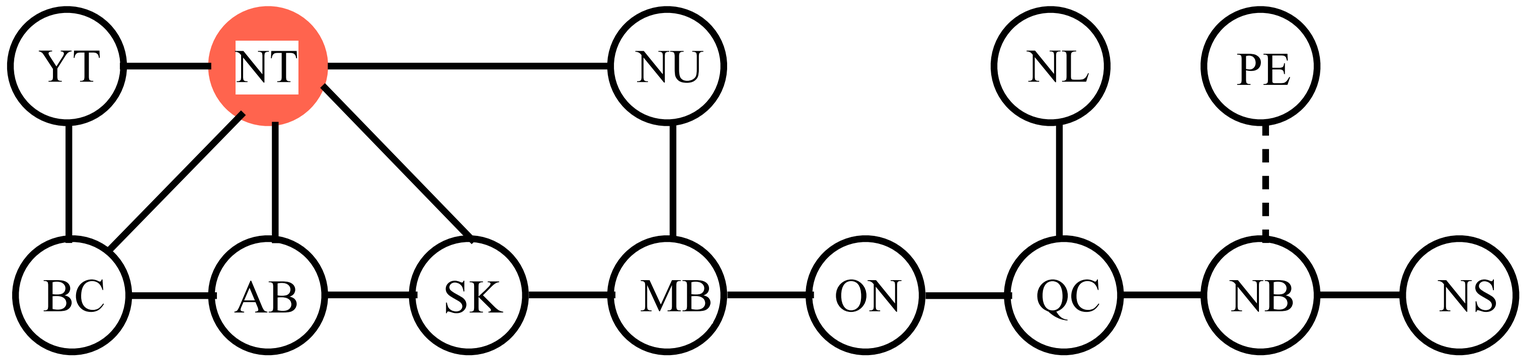
Choose to color NT: only 1 choice possible while e.g. AB has two.

Degree Heuristics

MRV can speed-up the process significantly (although depending on the problem).
What it cannot do is to decide with which variable is better to start or breaking ties.

A possibility is to use the degree heuristics:
Choose the variable involved in the largest number of constraints.
The idea is that this choice will maximise the reduction of the future possibilities.

Recalling the example of the "Canada graph", NT is the province connected with most constraints.



Value Ordering

Once a variable ordering algorithm is in place, we need to order the values for a variable.
The least-constraining-value heuristic is an efficient choice:

This strategy chooses the value (the color, in the map example) which rules out the fewer choices for the neighbouring variables in the constraint graph. In other words: it is the choice which leaves maximum flexibility.

If e color SK with blue, we constrain NT too much.
Actually in this case no color is allowed at all.

Forward Checking, or Ruling out Failures early on

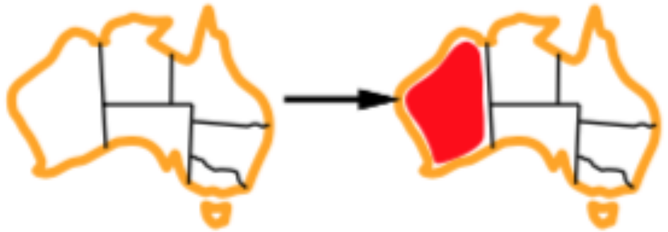
Idea: apply inference during the search.
Remember: algorithms like AC3 should be run instead before the search.

Simplest algorithm: forward checking
Once a variable X is chosen, FC checks for arc-consistency, i.e. for every (unassigned) variable Y connected to X via a constraint, delete from D_Y the values incompatible with the value chosen for X.

Forward Checking 1

Considering for simplicity a “smaller” country, we follow BACKTRACKING with FC:

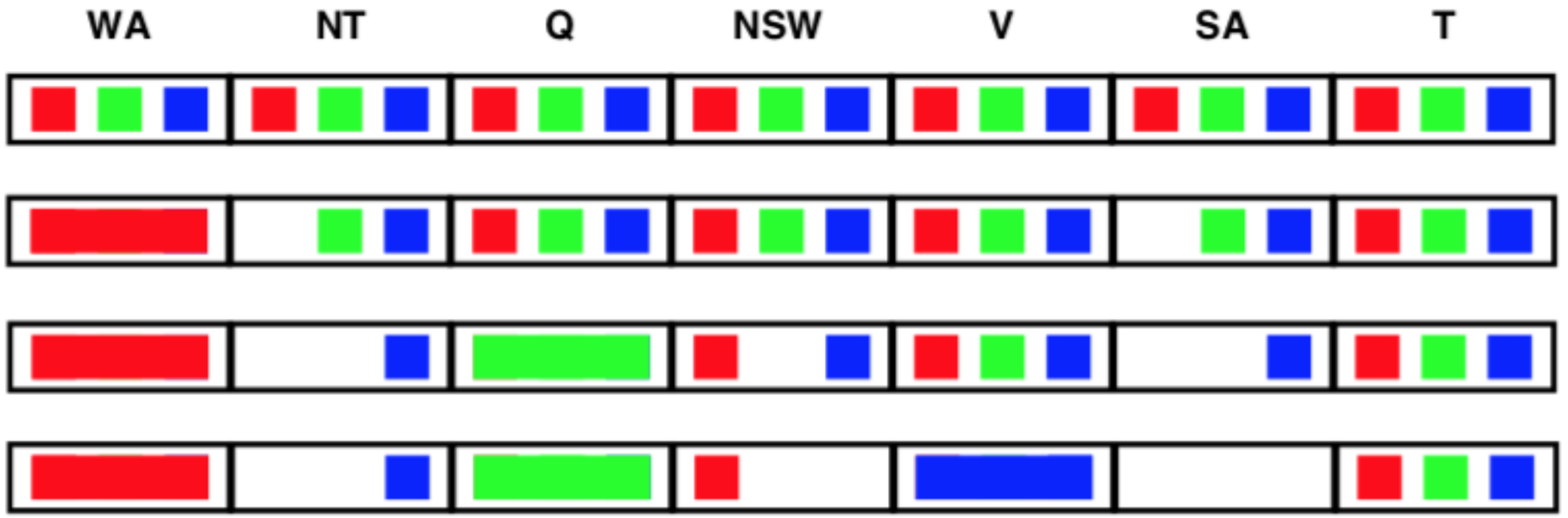
Forward Checking 2



Forward Checking 3

Two variable domains (NT, SA) reduced to.1 choice.

Forward Checking 4



One domain (SA) is now empty! Backtracking needed…

Some considerations remembering AC3

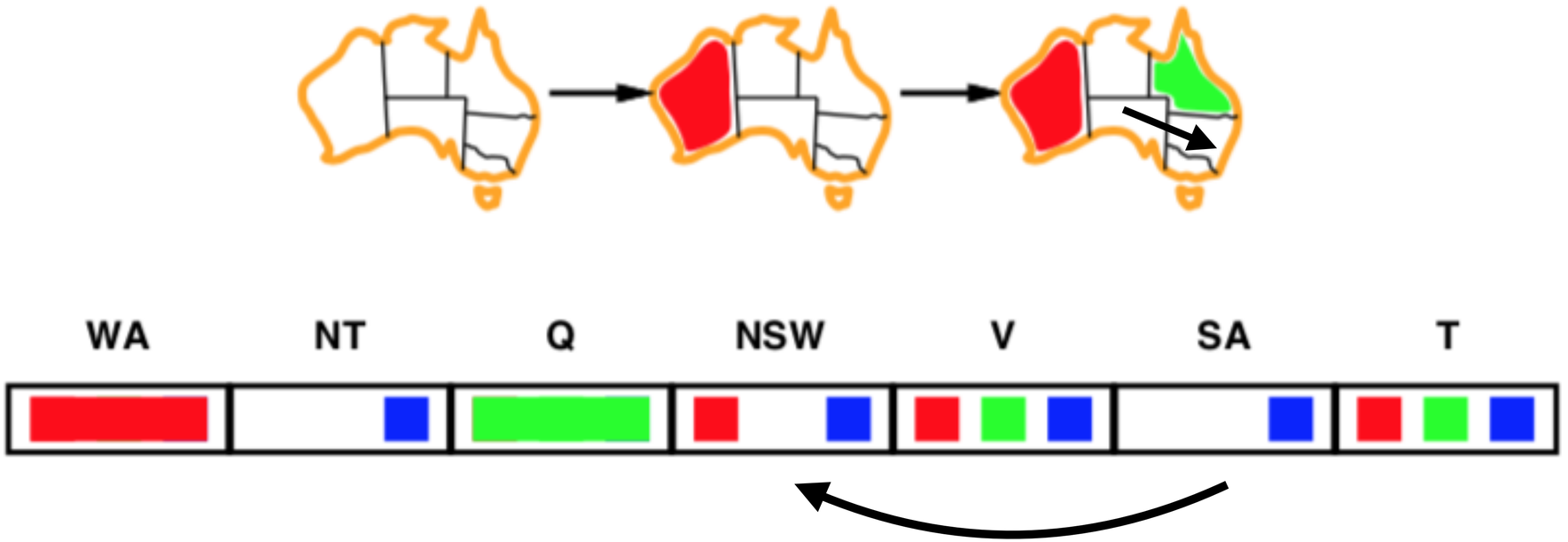
FC propagates information from assigned variables to unassigned variables.
Therefore: no information exchange between unassigned variables.

Remember arc-consistency:

- A directed arc X → Y is "consistent" iff for every value x of X, there exists a value y of Y, such that (x, y) satisfies the constraint between X and Y.

- Remove values from the domain of X to enforce arc-consistency.
- Arc consistency detects failures earlier.
- Can be used as preprocessing technique or as a propagation step during backtracking.

Arc-consistency: example

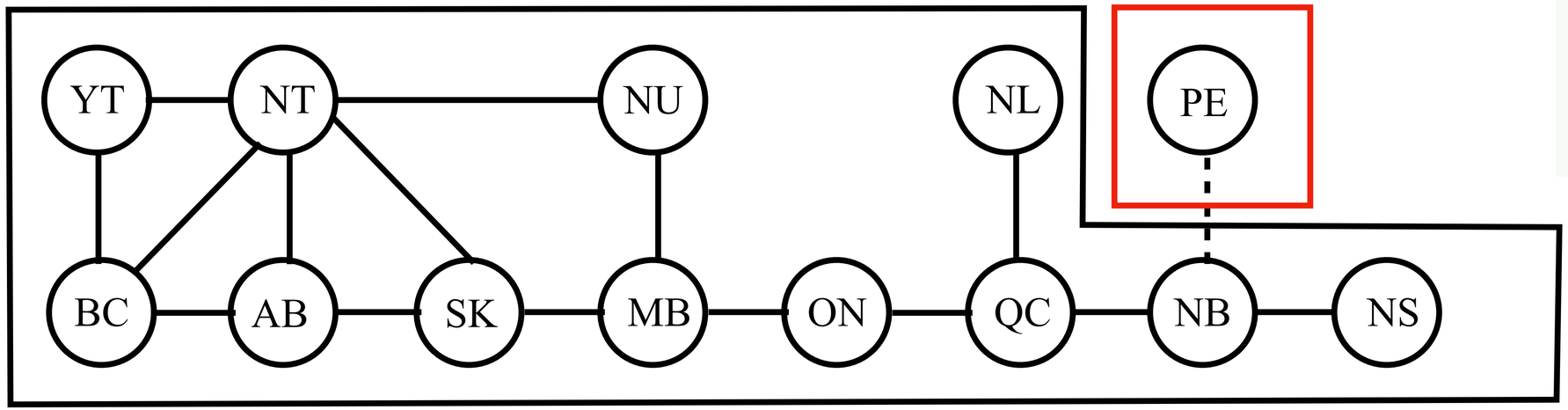


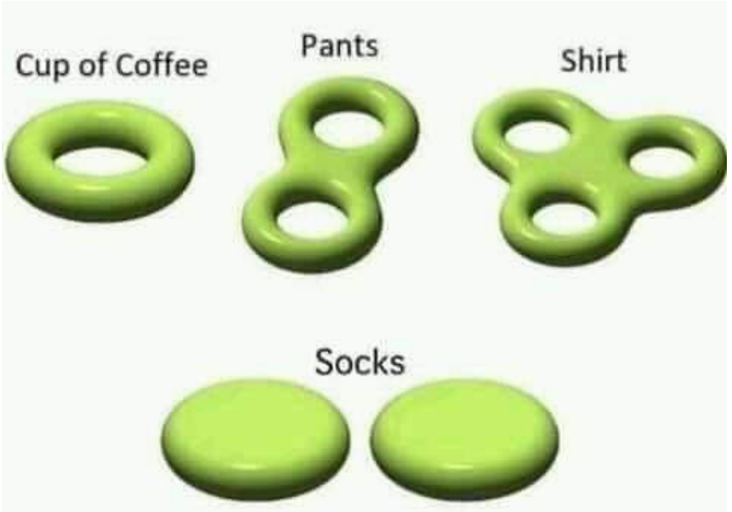
Arc-consistency between SA and NSW implies the removal of “blue” from NSW

Arc-consistency between SA and NT leaves instead no assignments possible for NSW.

Structure and Topology 1

Understanding the structure (or the topology) of a problem can help in speeding up the search for a solution.
From the map example:

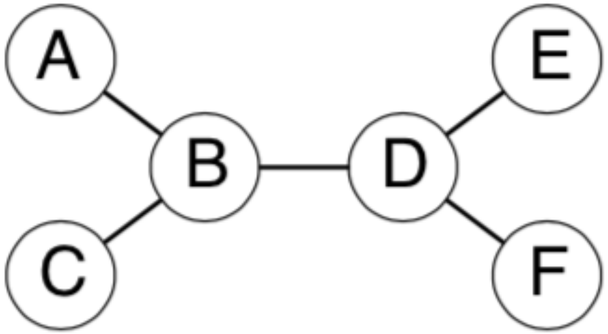




we have 2 disconnected graph components that can be solved separately.
This leads to 2 simpler, independent sub-problems.

Structure and Topology 2

Tree topology

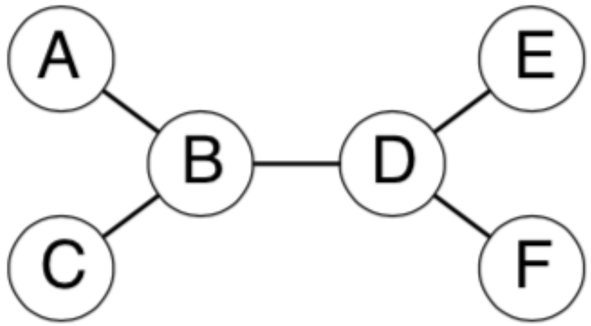


If the CSP graph of constraints is a tree (any 2 variables are connected only by 1 path), it can be solved in $O(nd^2)$ (linear in the number of variables!) while a general CSP needs $O(d^n)$ in the worst case.

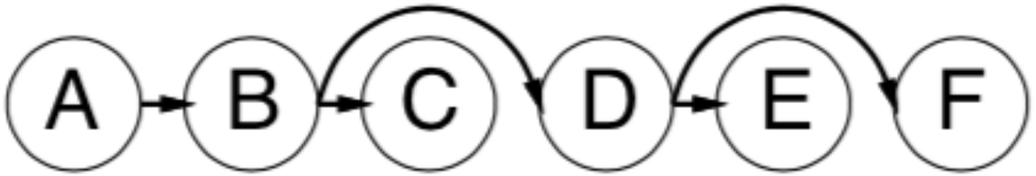
Idea of the algorithm:

- Order the nodes
- Apply directional arc-consistency (DAC) from leaves to root
- Assign values starting from the root.

Structure and Topology 3



topological variables sorting
(starting from A as root)



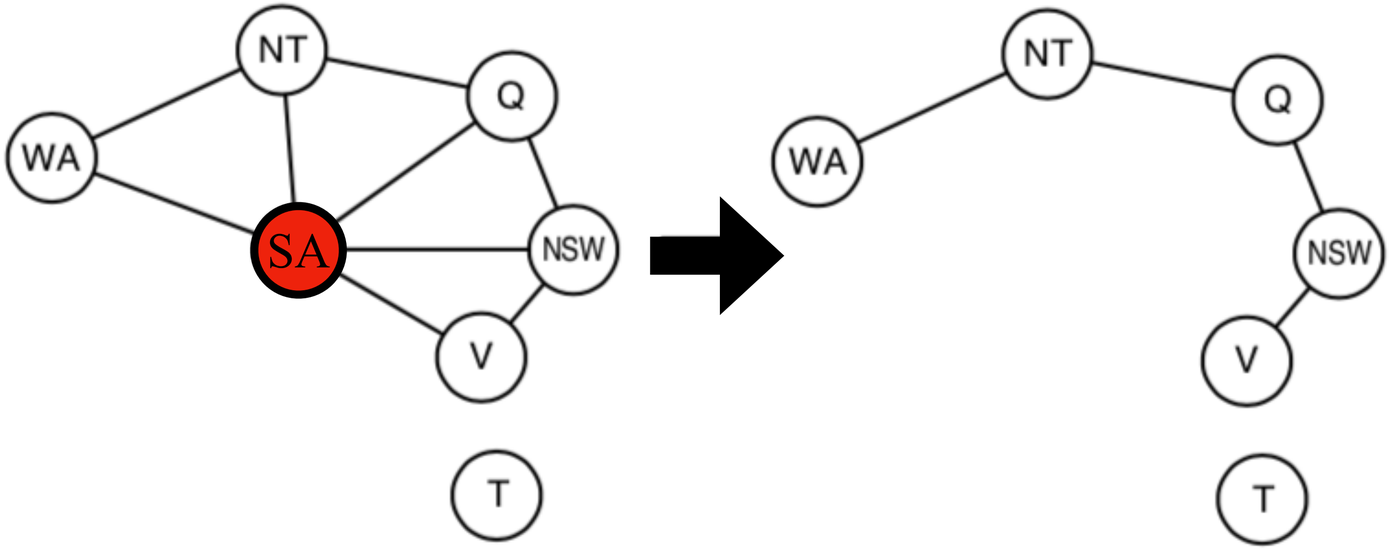
Directional arc-consistency: Given n variables under the ordering X_1, X_2, …, X_n, they have DAC iff every X_i is arc-consistent with X_j for j>i .

A tree with n nodes has (n-1) edges. We can enforce DAC in O(n) steps (going backwards). Each step must compare up to d values for couples of variables —> time complexity $O(nd^2)$.

Once we have DAC over the tree, we can start from the root and choose the remaining values. Since there is consistency, we know already that whatever value we pick, it will be consistent with one of the values of the next node.

No backtracking is needed!

Structure and Topology 4



Cutset conditioning: we can remove one node (here: SA) and turn the graph into a tree. We ca assign a value to SA and make the other nodes consistent to it. After that, we can run the previous linear algorithm.

Nodes like SA are called cycle cutsets.

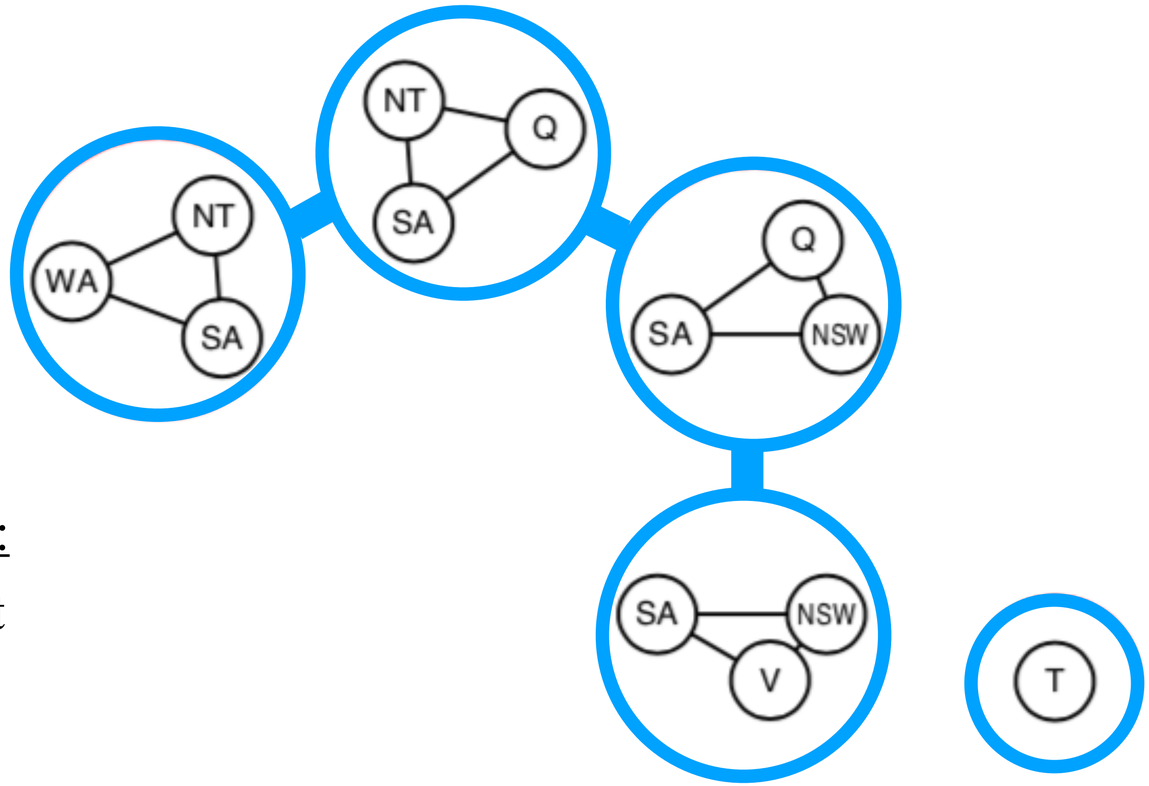
Finding cycle cutsets is unfortunately NP-hard, but approximate algorithms are known.

Structure and Topology 5

Reduce a graph to a tree: Tree Decomposition

- Decompose the problem into a set of connected sub-problems, where two sub-problems are connected when they share a constraint.

- Solve the sub-problems independently and then combine the solutions.



A tree decomposition must satisfy the following conditions:

1) Every variable of the original problem appears in at least one sub-problem.

2) Every constraint appears in at least one sub-problem

3) If a variable appears in two sub-problems, it must appear in all sub-problems on the path between the two sub-problems

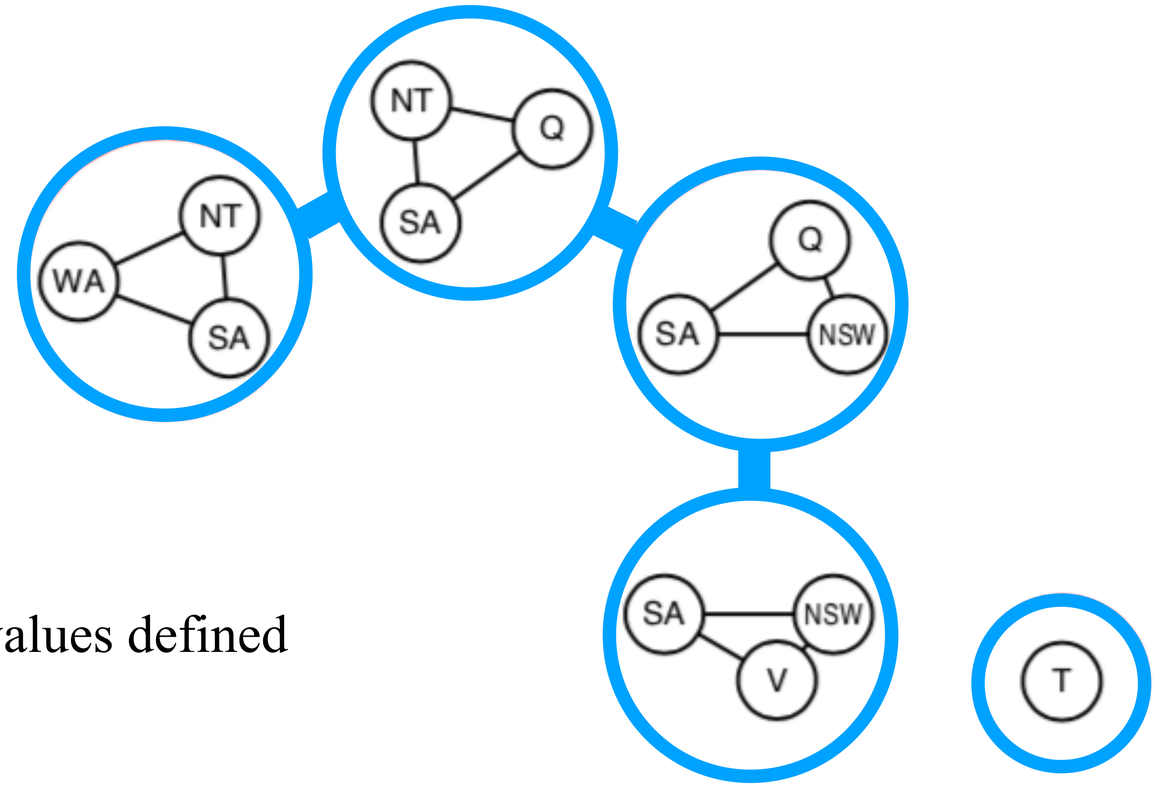
4) The connections form a tree

Meaning of the conditions:

1)+2) : all variables and constraints represented.

3) implies that any variable has the same value wherever it appeasers.

Structure and Topology 6



- Consider sub-problems as new collective variables, with values defined by the solutions to the sub-problems

- Use tree-CSP solver algorithm to find an overall solution

- Constraint: identical value for the same variable

Structure and Topology 7

- It is desirable to make all the subproblems as small as possible. The tree width of a tree decomposition is the size(largest sub-problem)-1

- Tree width of a graph is the minimal tree width over all possible tree decompositions

- If a graph has tree width **w** and we know a tree decomposition with that width, we can solve the problem in $O(nd^{w+1})$

- Unfortunately, finding a tree decomposition with minimal tree width is **NP**-hard. However, there are heuristic methods that work well in practice.

Summary

- CSPs represent states with variables/values pairs plus a set of constraints.
  These structures represent many real-world problems (scheduling, VLSI, …)

- Inference techniques can be used for reducing the number of variable values using the constraints.
  Examples are node-, arc-, path-, and k-consistency.

- Backtracking (depth-first search) is commonly used for solving CSPs and inference techniques are applied while searching.

- Heuristics like MRV can be used for deciding which variable to use next during backtracking.
  Least-constraining-value heuristics can be used for deciding which value to try first.

- Cutset conditioning and tree decomposition are two ways to transform part of the problem into a tree
  CSPs with tree topology can also be solved using local search