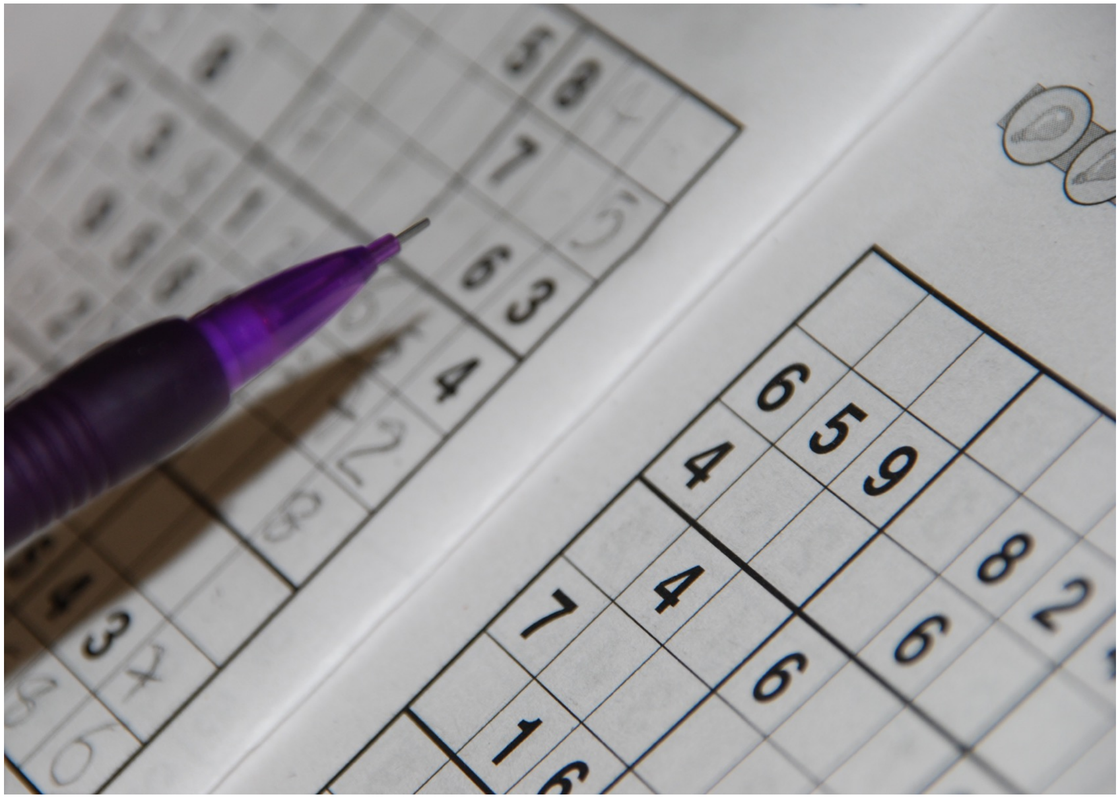
Artificial Intelligence

Constraint Satisfaction Problems



Recall

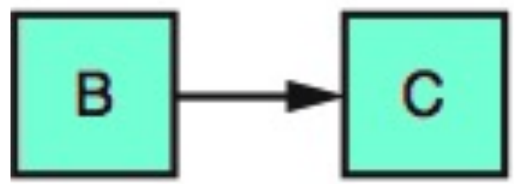
- **Search problems:**
  - Find the **sequence of actions** that leads to the goal.
  - Sequence of actions means a **path** in the search space.
  - Paths come with different costs and depths.
  - We use "rules of thumb" aka **heuristics** to guide the search efficiently.

Recall

- **Search problems:**
  - Find the **sequence of actions** that leads to the goal.
  - Sequence of actions means a **path** in the search space.
  - Paths come with different costs and depths.
  - We use "rules of thumb" aka **heuristics** to guide the search efficiently.
- **Constraint satisfaction problems:**
  - A search problem too!
  - We care about the **goal itself**.

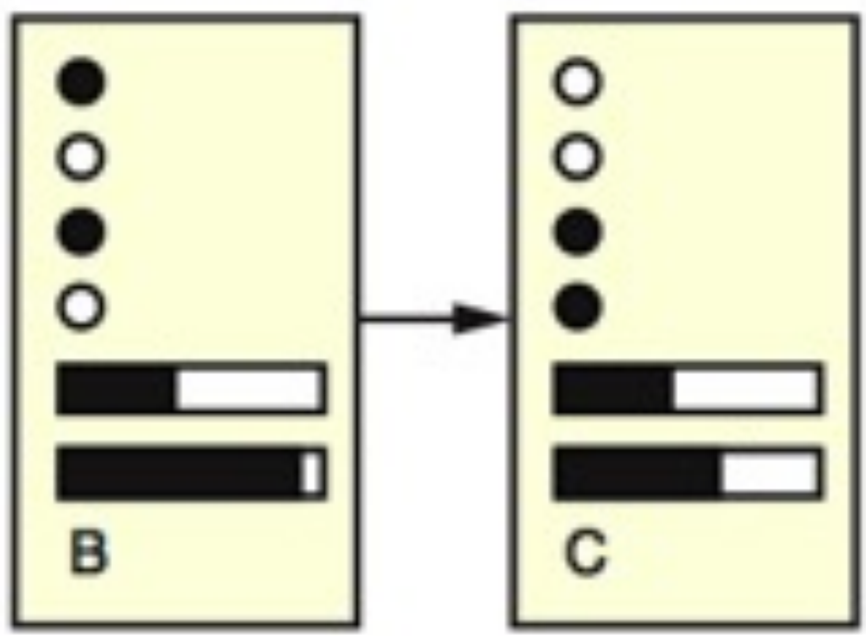
CSPs definition

- **Search problems:**
  - A state is a **black box**, implemented as some data structure. Recall atomic representation.
  - A goal test is a function over the states.

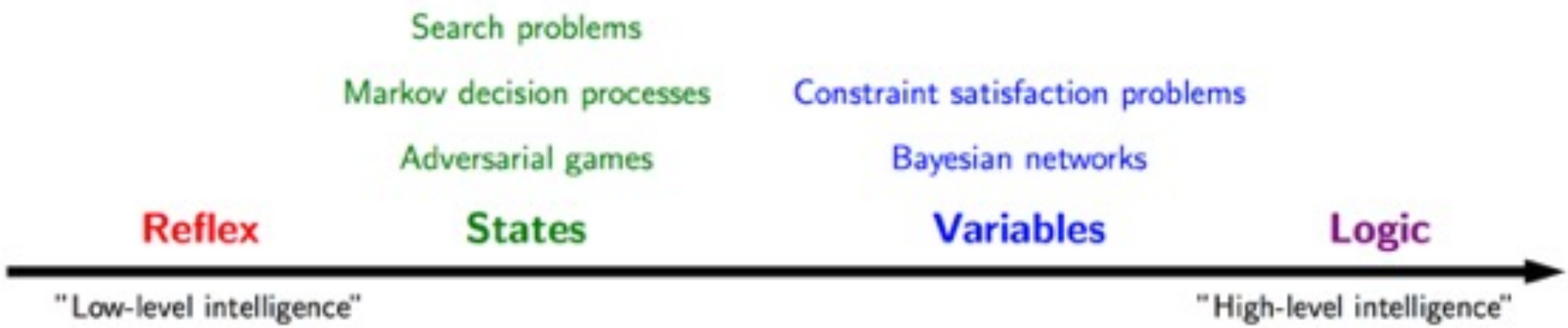


CSPs definition

- **CSPs problems:**
  - A state: defined by variables X_i with values from domain D_i. Recall factored representation.
  - A goal test is a **set of constraints** specifying **allowable combinations** of values for subsets of variables.



CSPs definition



Credit: Courtesy Percy Liang

CSPs definition

- A constraint satisfaction problem consists of **three elements**:
  - A set of **variables**, $X = \{X_1, X_2, \cdots X_n\}$
  - A set of **domains** for each variable: $D = \{D_1, D_2, \cdots D_n\}$
  - A set of **constraints** C that specify allowable combinations of values.

CSPs definition

- A constraint satisfaction problem consists of **three elements**:
  - A set of **variables**, $X = \{X_1, X_2, \cdots X_n\}$
  - A set of **domains** for each variable: $D = \{D_1, D_2, \cdots D_n\}$
  - A set of **constraints** C that specify allowable combinations of values.
- Solving the CSP: **finding the assignment(s)** that **satisfy all constraints**.
- Concepts: problem formalization, backtracking search, arc consistency, etc.

CSPs definition

- A constraint satisfaction problem consists of **three elements**:
  - A set of **variables**, $X = \{X_1, X_2, \cdots X_n\}$
  - A set of **domains** for each variable: $D = \{D_1, D_2, \cdots D_n\}$
  - A set of **constraints** C that specify allowable combinations of values.
- Solving the CSP: **finding the assignment(s)** that **satisfy all constraints**.
- Concepts: problem formalization, backtracking search, arc consistency, etc.
- We call a solution, a **consistent assignment**.

Example: Map coloring

Variables: $X = \{WA, NT, Q, NSW, V, SA, T\}$

Example: Map coloring

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

Domains: D_i = {red, green, blue}

Example: Map coloring

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

Domains: D_i = {red, green, blue}

Constraints: adjacent regions must have different colors;

Example: Map coloring

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

Domains: D_i = {red, green, blue}

Constraints: adjacent regions must have different colors;

e.g., WA $\neq$ NT or (W A, N T) $\in$ {(red, green), (red, blue), etc..}

Example: Map coloring

Example: Map coloring

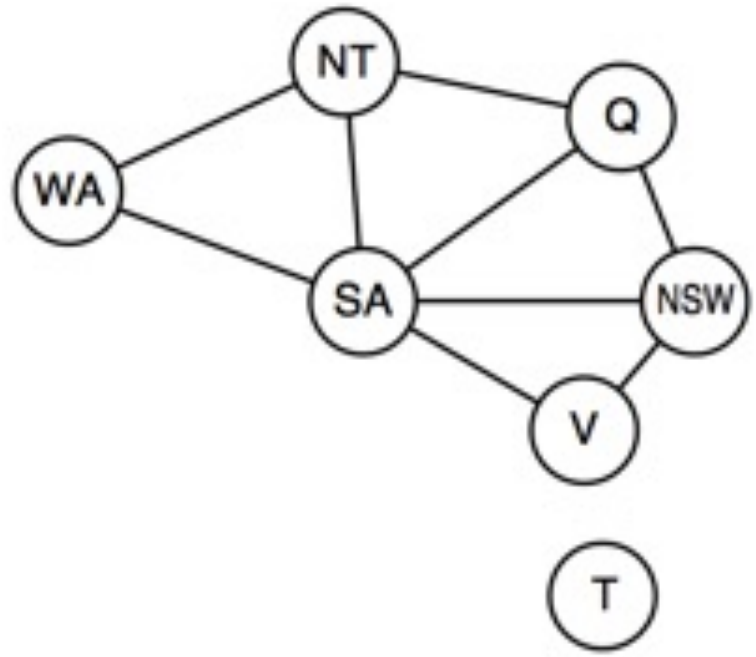
Example:

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

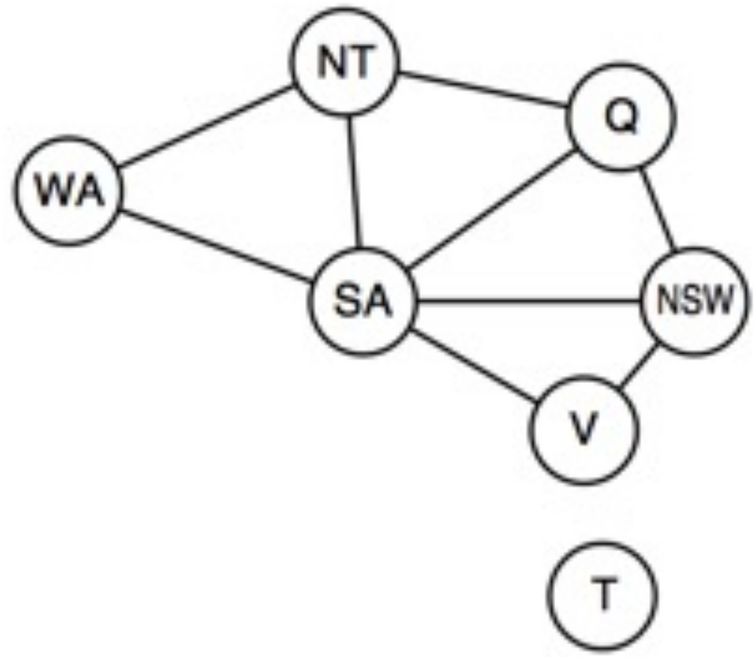
Real-world CSPs

- Assignment problems, e.g., who teaches what class?
- Timetabling problems, e.g., which class is offered when and where?
- Hardware configuration
- Spreadsheets
- Transportation scheduling
- Factory scheduling
- Floor planning
- Notice that many real-world problems involve real-valued variables

Constraint graph

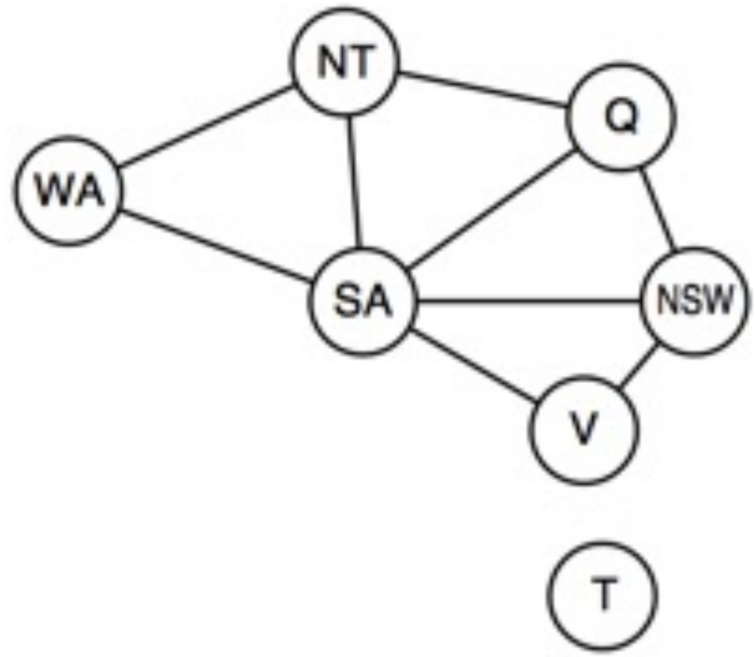


Constraint graph



Binary CSP: each constraint relates at most two variables Constraint graph: nodes are variables, arcs show constraints

Constraint graph



Binary CSP: each constraint relates at most two variables Constraint graph: nodes are variables, arcs show constraints

CSP algorithms: use the graph structure to speed up search. E.g., Tasmania is an independent subproblem!

Varieties of variables

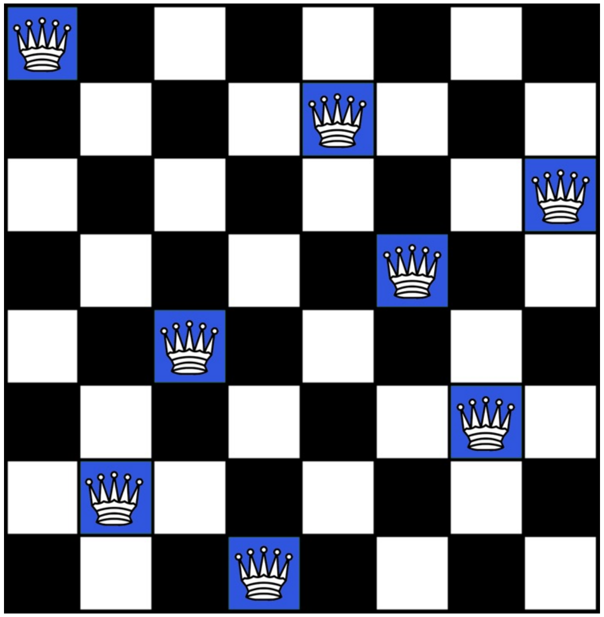
- **Discrete variables**:
  - Finite domains:
    * assume n variables, d values, then the number of complete assignments is $O(d^n)$.
    * e.g., map coloring, 8-queens problem
  - Infinite domains (integers, strings, etc.):
    * need to use a constraint language,
    * e.g., job scheduling. $T_1 + d \leq T_2$.
- **Continuous variables**:
  - Common in operations research
  - Linear programming problems with linear or non linear equalities

Varieties of constraints

- **Unary constraints**: involve a single variable e.g., SA $\neq$ green
- **Binary constraints**: involve pairs of variables e.g., SA $\neq$ WA
- **Global constraints**: involve 3 or more variables e.g., *Alldiff* that specifies that all variables must have different values (e.g., cryptarithmetic puzzles, Sudoku)
- **Preferences (soft constraints)**:
  - Example: red is better than green
  - Often represented by a cost for each variable assignment
  - constrained optimization problems

Example: 8-queen

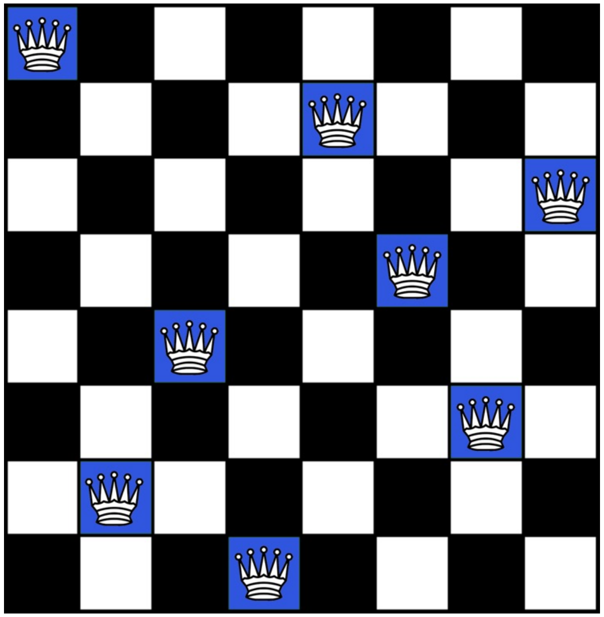
8-Queen: Place 8 queens on an 8x8 chess board so no queen can attack another one.



Problem formalization:

Example: 8-queen

8-Queen: Place 8 queens on an 8x8 chess board so no queen can attack another one.

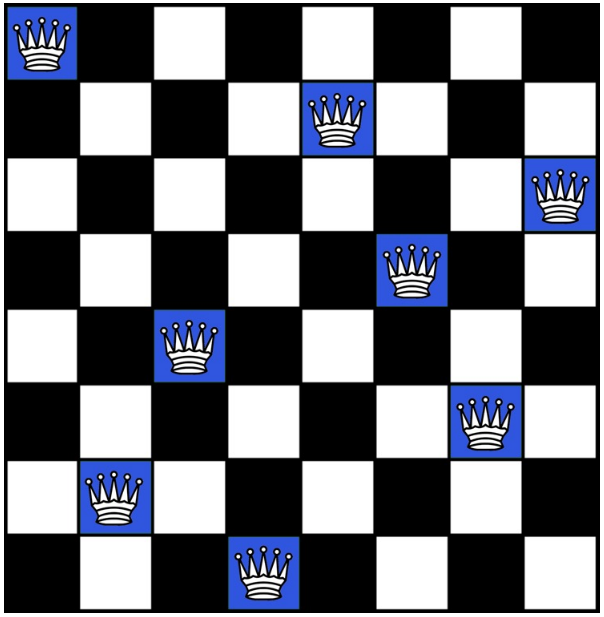


Problem formalization 1:

- One variable per queen, Q_1, Q_2, . . . , Q_8.
- Each variable could have a value between 1 and 64.
- Solution: $Q_1 = 1$, $Q_2 = 13$, $Q_3 = 24$, . . . , $Q_8 = 60$.

Example: 8-queen

8-Queen: Place 8 queens on an 8x8 chess board so no queen can attack another one.

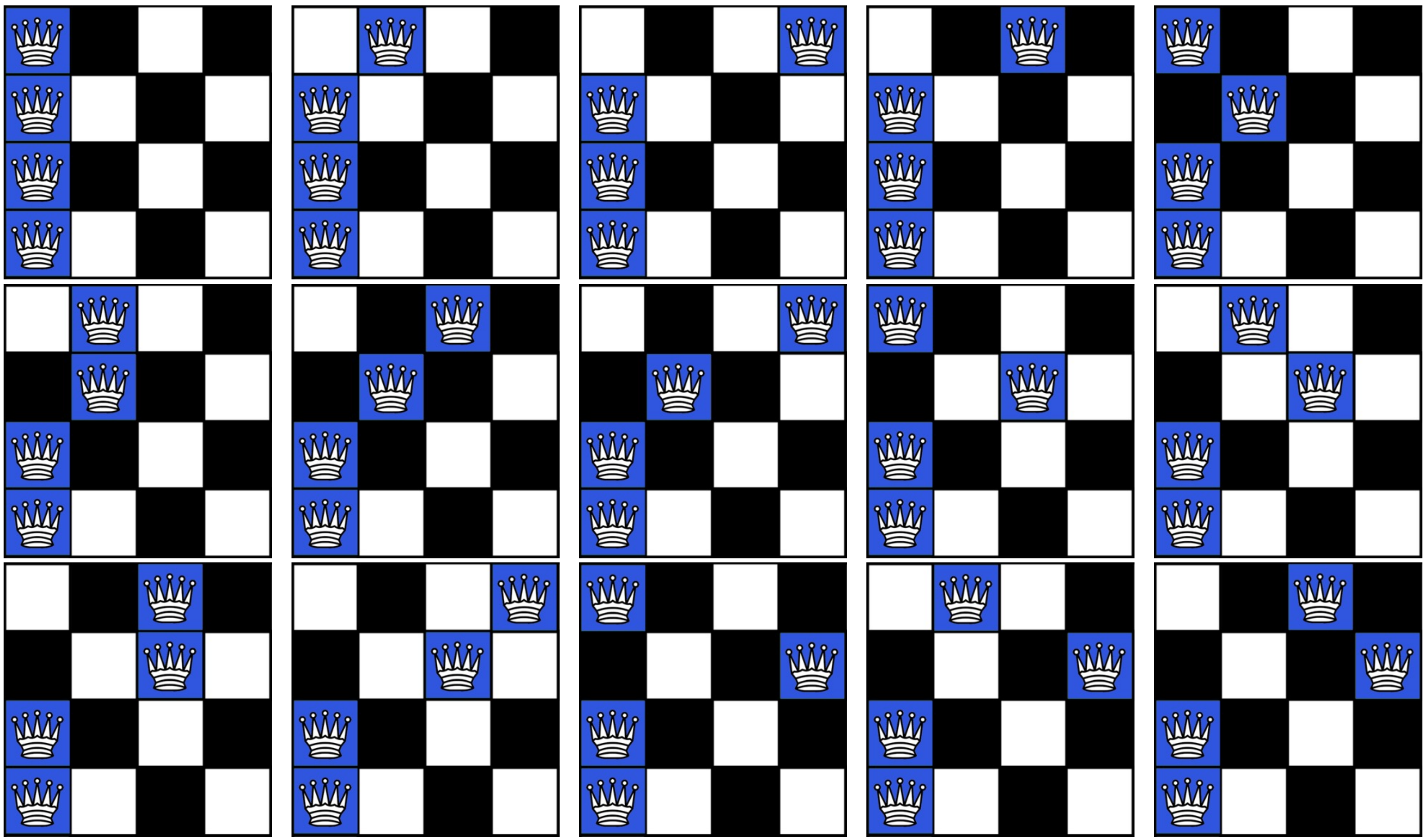


Problem formalization 2:

- One variable per queen, Q_1, Q_2, . . . , Q_8.
- Each variable could have a value between 1 and 8 (columns).
- Solution: $Q_1 = 1$, $Q_2 = 7$, $Q_3 = 5$, . . . , $Q_8 = 3$.

Brute force?

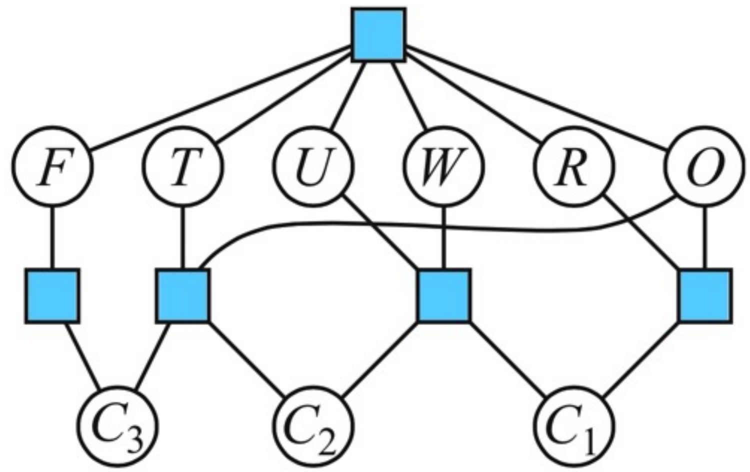
Should we simply generate and test all configurations?



. . .

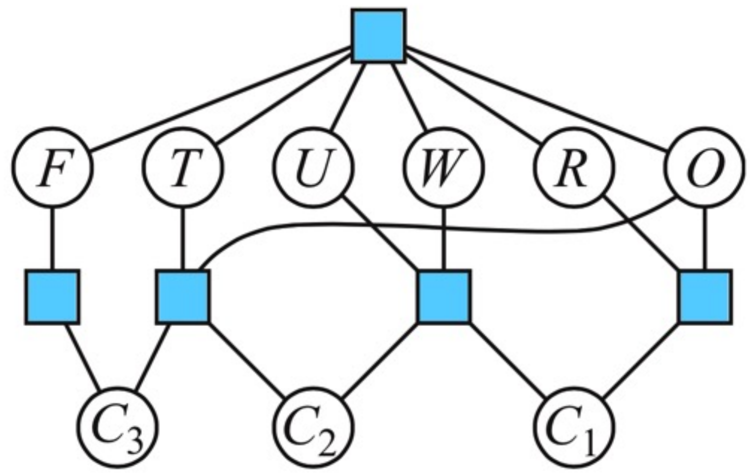
Example Cryptarithmetic

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



Example Cryptarithmetic

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



Variables: $X = \{F, T, U, W, R, O, C_1, C_2, C_3\}$

Domain: $D = \{0, 1, 2, \cdots, 9\}$

Constraints:

- Alldiff(F, T, U, W, R, O)
- $T \neq 0, \;\; F \neq 0$
- $O + O = R + 10 * C_1$
- $C_1 + W + W = U + 10 * C_2$
- $C_2 + T + T = O + 10 * C_3$
- $C_3 = F$

Solving CSPs

- **State-space search algorithms: search!**
- **CSP Algorithms:** Algorithm can do two things:
  - **Search**: choose a new variable assignment from many possibilities
  - **Inference**: constraint propagation, use the constraints to spread the word: reduce the number of values for a variable which will reduce the legal values of other variables etc.
- As a preprocessing step, constraint propagation can sometimes solve the problem entirely without search.
- Constraint propagation can be intertwined with search.

Solving CSPs

- **BFS:** Develop the complete tree
- **DFS:** Fine but time consuming
- **BTS: Backtracking search** is the basic uninformed search for CSPs. It's a DFS s.t.

  1. Assign one variable at a time: assignments are commutative. e.g., (WA=red, NT=green) is same as (NT=green, WA=red)

  2. Check constraints on the go: consider values that do not conflict with previous assignments.

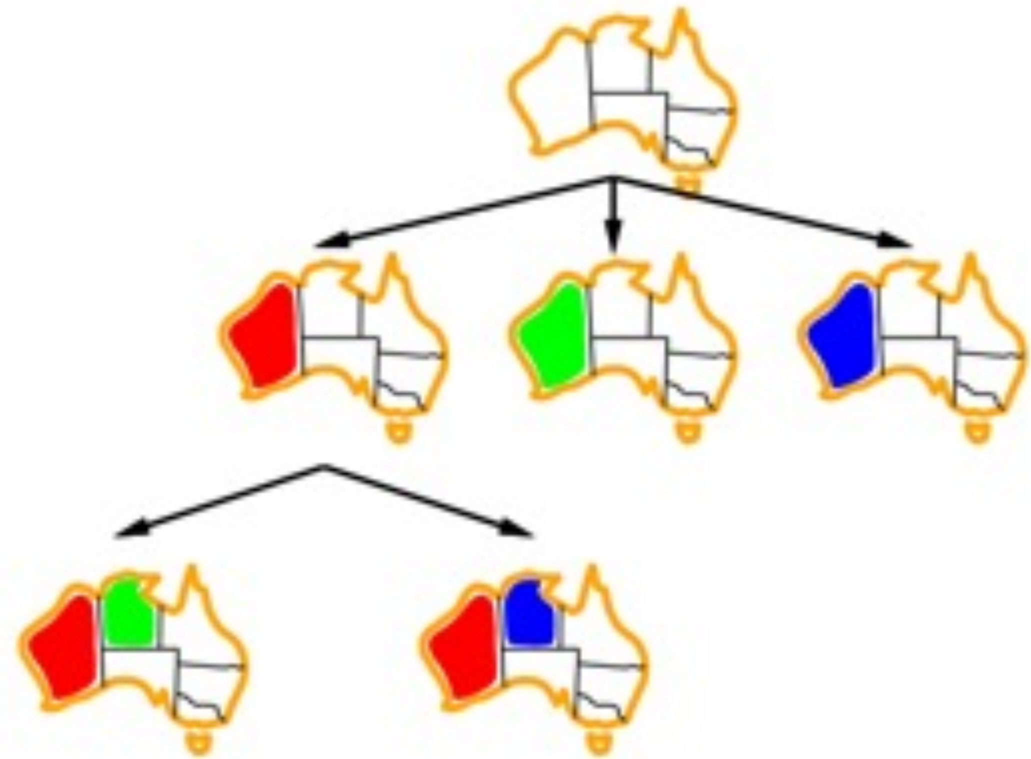
Solving CSPs

- **Initial state**: empty assignment $\{\}$
- **States**: are partial assignments
- **Successor function**: assign a value to an unassigned variable
- **Goal test**: the current assignment is complete and satisfies all constraints

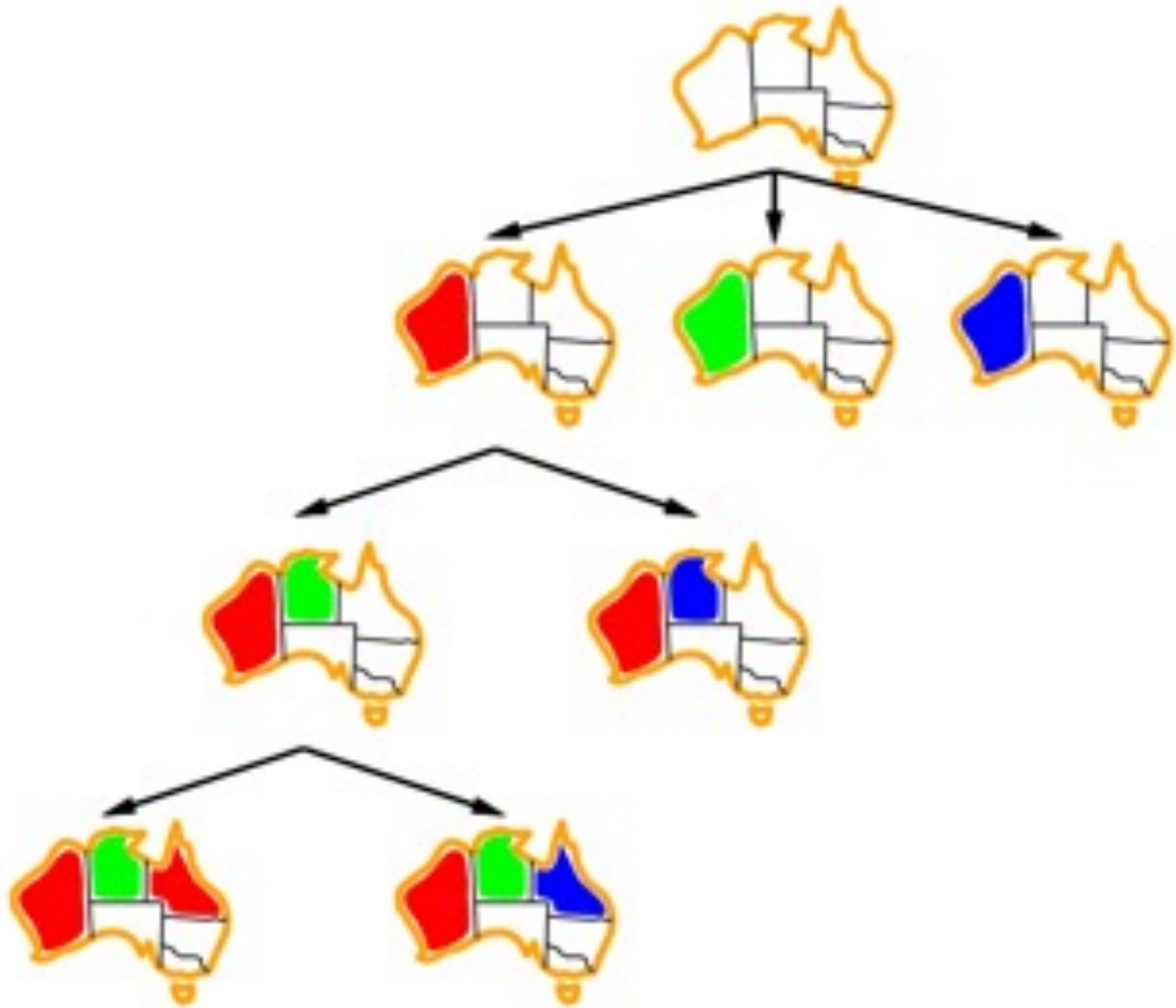
Backtracking search

Backtracking search

Backtracking search



Backtracking search



Improving BTS

Heuristics are back!

1. **Which variable should be assigned next?**

Improving BTS

Heuristics are back!

1. **Which variable should be assigned next?**
2. **In what order should its values be tried?**

Improving BTS

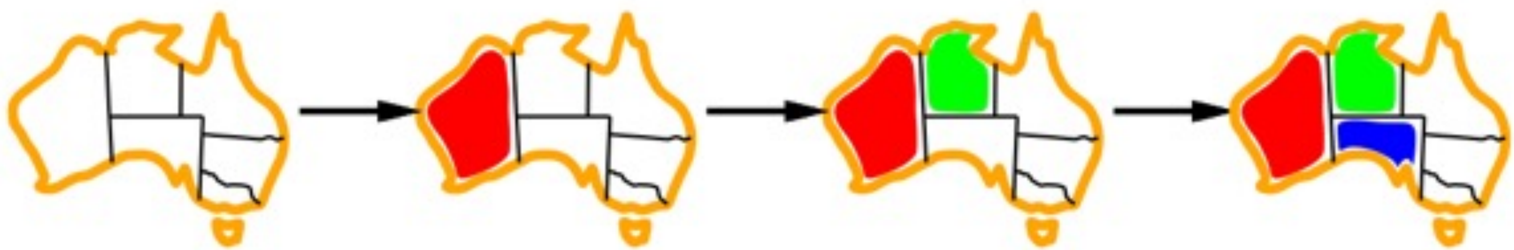
Heuristics are back!

1. Which variable should be assigned next?
2. In what order should its values be tried?
3. Can we detect inevitable failure early?

Minimum Remaining Values

1. Which variable should be assigned next?

- **MRV**: Choose the variable with the fewest legal values in its domain

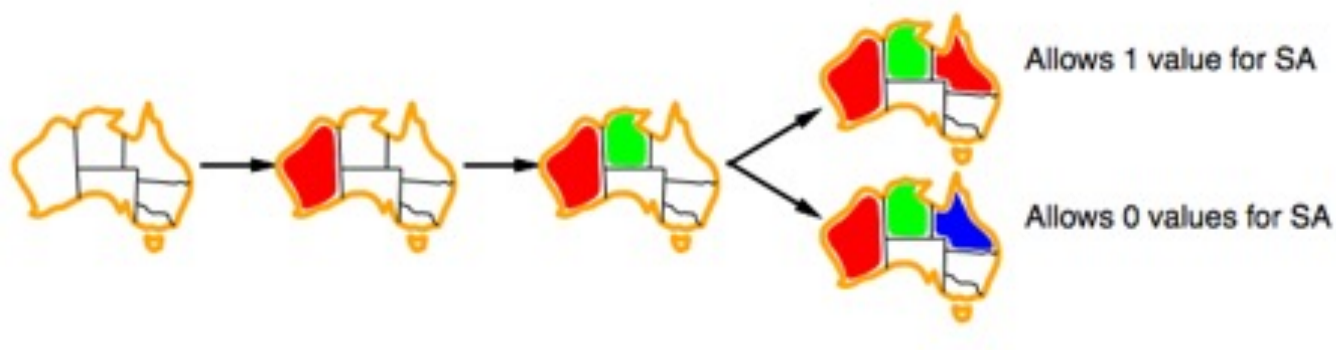


Pick the hardest!

Least constraining value

2. In what order should its values be tried?

- **LCV**: Given a variable, choose the least constraining value: the one that rules out the fewest values in the remaining variables



Pick the ones that are likely to work!

Forward checking

3. Can we detect inevitable failure early?

- FC: Keep track of remaining legal values for the unassigned variables. Terminate when any variable has no legal values.

Forward checking

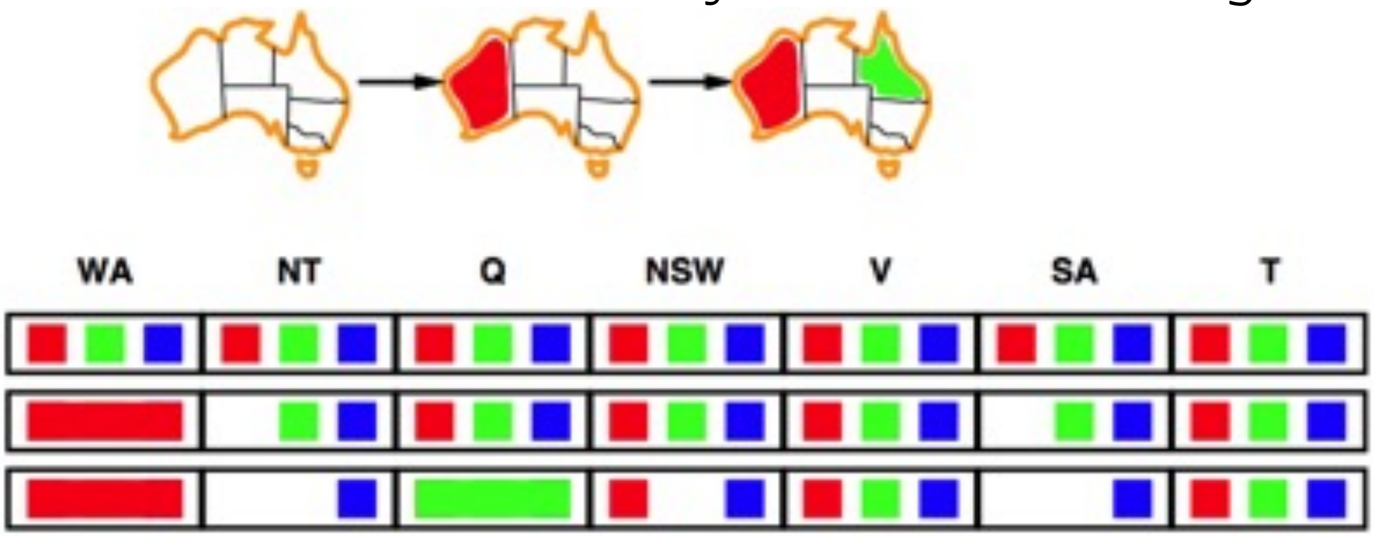
3. Can we detect inevitable failure early?

- **FC**: Keep track of remaining legal values for the unassigned variables. Terminate when any variable has no legal values.

Forward checking

3. Can we detect inevitable failure early?

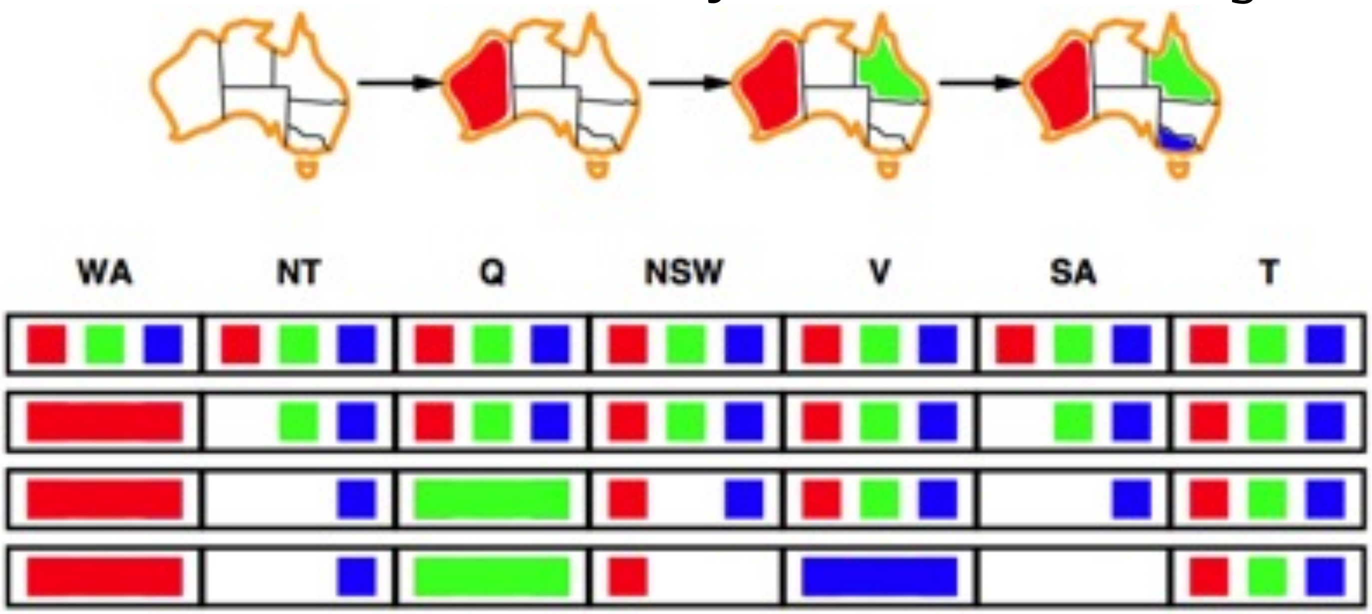
- FC: Keep track of remaining legal values for the unassigned variables. Terminate when any variable has no legal values.



Forward checking

3. Can we detect inevitable failure early?

- **FC**: Keep track of remaining legal values for the unassigned variables. Terminate when any variable has no legal values.



Backtracking search

```
function BACKTRACKING_SEARCH(csp) returns a solution, or failure
    return BACKTRACK({}, csp)

function BACKTRACK(assignment, csp) returns a solution, or failure
    if assignment is complete then return assignment
    var = SELECT_UNASSIGNED-VARIABLE(csp)
    for each value in ORDER_DOMAIN_VALUES (var, assignment, csp)
        if value is consistent with assignment then
            add {var = value} to assignment
            result = BACKTRACK(assignment, csp)
            if result ≠ failure then return result
        remove {var = value} from assignment
    return failure
```

Solving CSPs: Sudoku

All 3x3 boxes, rows, columns, must contain all digits 1..9.

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

Solving CSPs: Sudoku

All 3x3 boxes, rows, columns, must contain all digits 1..9.

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

Variables: $V = \{A_1, \cdots, A_9, B_1, \cdots, B_9, \cdots, I_1 \cdots I_9\}$, $|V| = 81$.

Domain: $D = \{1, 2, \cdots, 9\}$, the filled squares have a single value.

Constraints: 27 constraints

- Alldiff$(A_1, A_2, A_3, A_4, A_5, A_6, A_7, A_8, A_9)$

  ...
- Alldiff$(A_1, B_1, C_1, D_1, E_1, F_1, G_1, H_1, I_1)$

  ...
- Alldiff$(A_1, A_2, A_3, B_1, B_2, B_3, C_1, C_2, C_3)$

Solving CSPs: Sudoku

All 3x3 boxes, rows, columns, must contain all digits 1..9.

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

Solving CSPs: Sudoku

All 3x3 boxes, rows, columns, must contain all digits 1..9.

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

- Naked doubles (triples): find two (three) cells in a 3x3 grid that have only the same candidates left, eliminate these two (three) values from all possible assignments in that box.
- Locked pair, Locked triples, etc.

Solving CSPs: Sudoku

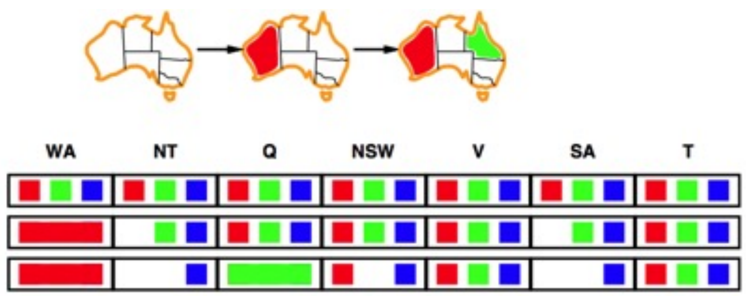
All 3x3 boxes, rows, columns, must contain all digits 1..9.

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

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

Constraint propagation

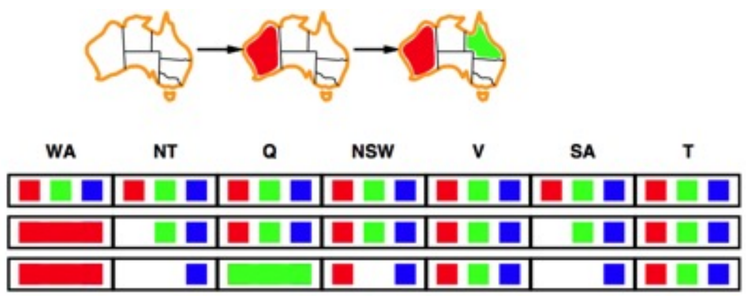
- Forward checking propagates information from assigned to unassigned variables.
- Observe:



  - Forward checking does not check interaction between unassigned variables! Here SA and NT! (They both must be blue but can't be blue!).

Constraint propagation

- Forward checking propagates information from assigned to unassigned variables.
- Observe:



  – Forward checking does not check interaction between unassigned variables! Here SA and NT! (They both must be blue but can't be blue!).

- Forward checking improves backtracking search but does not look very far in the future, hence does not detect all failures.
- We use constraint propagation, reasoning from constraint to constraint. e.g., arc consistency test.

Types of Consistency

- **Node-consistency** (unary constraints): A variable X_i is **node-consistent** if all the values of $Domain(X_i)$ satisfy all unary constraints.

Types of Consistency

- **Node-consistency** (unary constraints): A variable X_i is **node-consistent** if all the values of $Domain(X_i)$ satisfy all unary constraints.
- **Arc-consistency** (binary constraints): $X \rightarrow Y$ is arc-consistent if and only if every value x of X is consistent with some value y of Y.

Types of Consistency

- **Node-consistency** (unary constraints): A variable X_i is **node-consistent** if all the values of $Domain(X_i)$ satisfy all unary constraints.

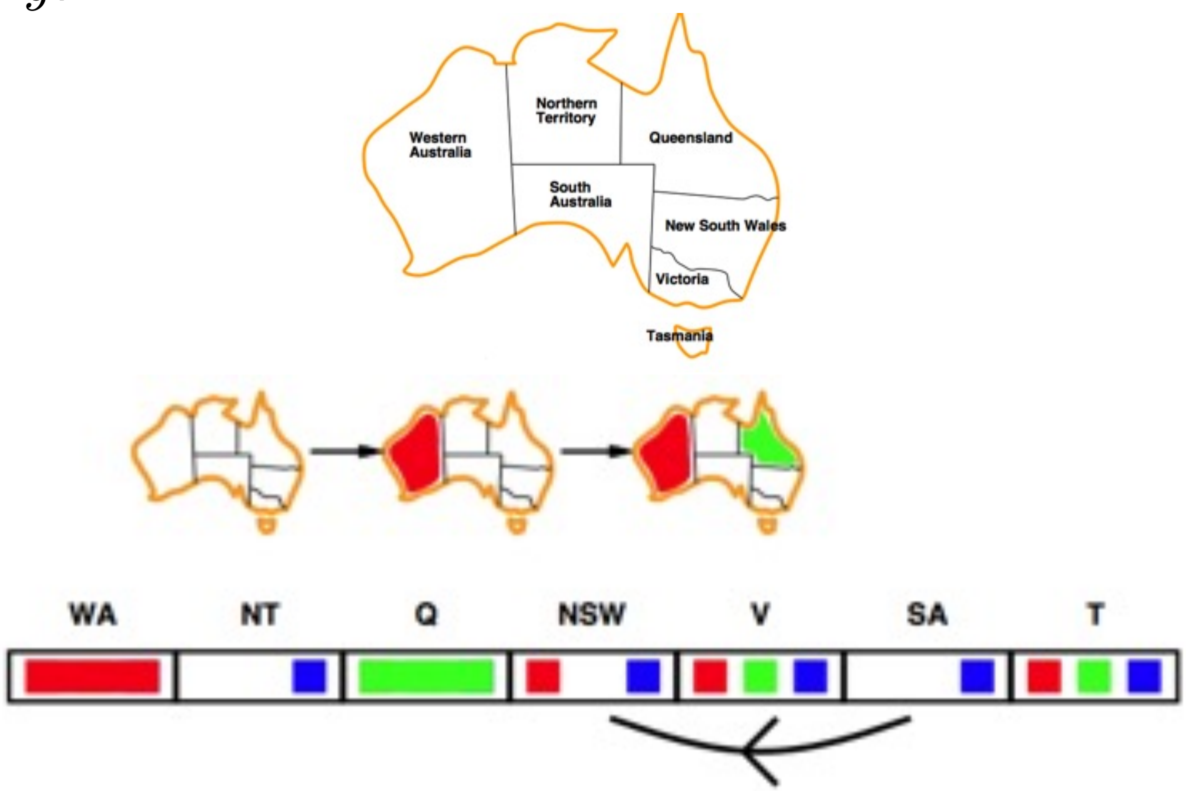
- **Arc-consistency** (binary constraints): $X \rightarrow Y$ is arc-consistent if and only if every value x of X is consistent with some value y of Y.

- **Path-consistency** (n-ary constraints): generalizes arc-consistency from binary to multiple constraints.

- **Note:** It is always possible to transform all n-ary constraints into binary constraints. Often, CSPs solvers are designed to work with binary constraints.

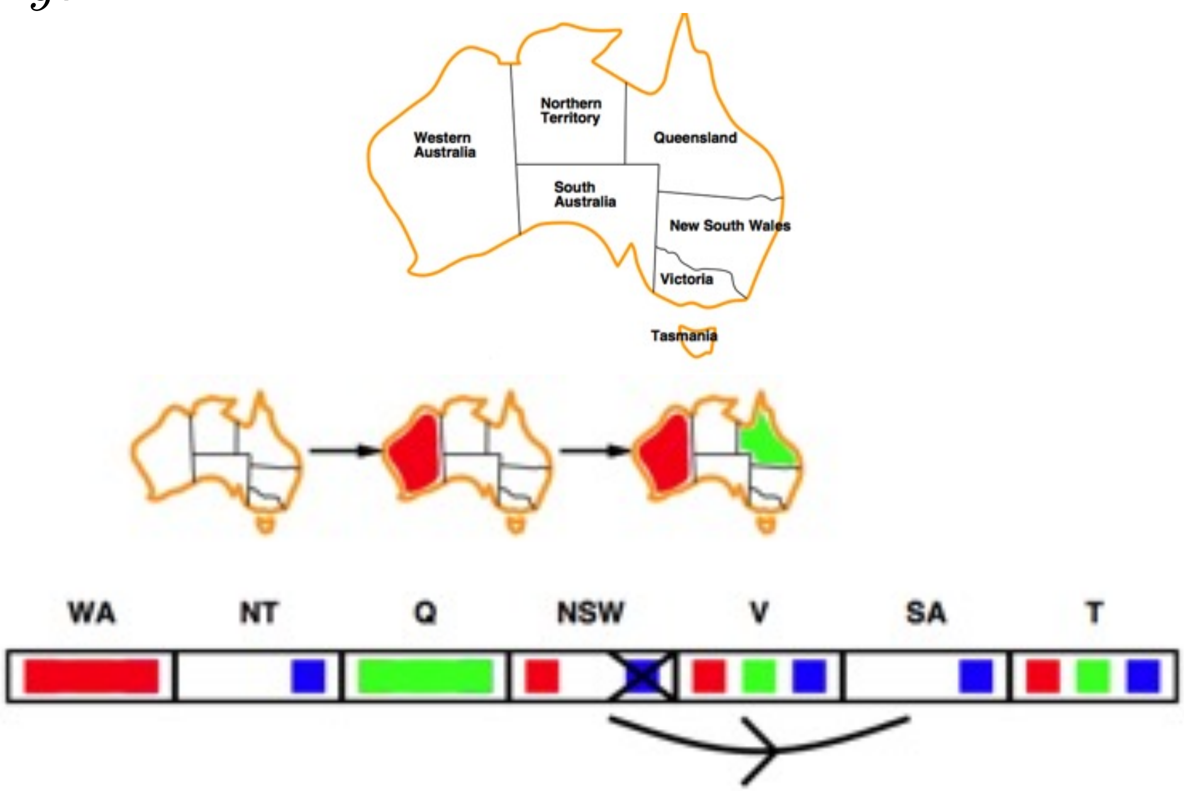
Arc consistency

- **AC**: Simplest form of propagation makes each arc consistent.
- $X \rightarrow Y$ is consistent IFF for every value x of X, there is some allowed y.



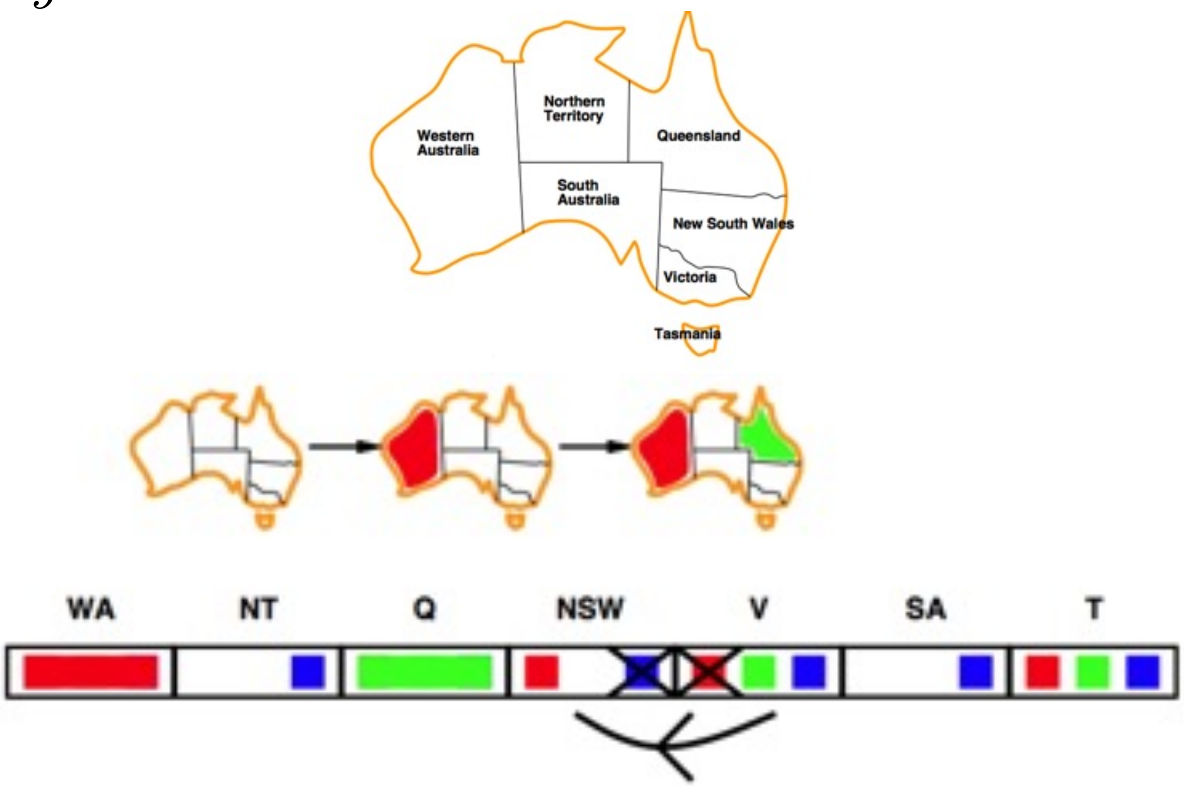
Arc consistency

- **AC**: Simplest form of propagation makes each arc consistent.
- $X \to Y$ is consistent IFF for every value x of X, there is some allowed y.



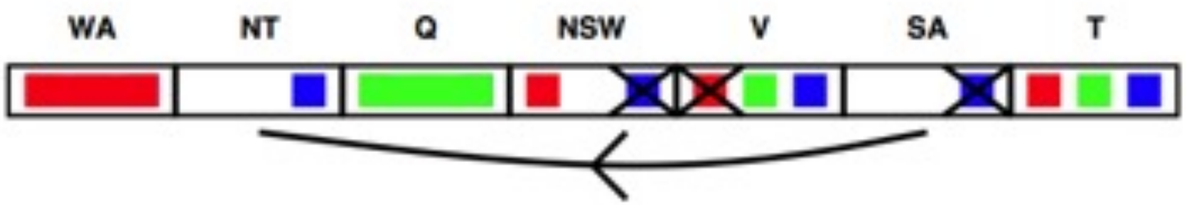
Arc consistency

- **AC**: Simplest form of propagation makes each arc consistent.
- $X \rightarrow Y$ is consistent IFF for every value x of X, there is some allowed y.



Arc consistency

- **AC**: Simplest form of propagation makes each arc consistent.
- $X \to Y$ is consistent IFF for every value x of X, there is some allowed y.



Arc consistency

Algorithm that makes a CSP arc-consistent!

```
function AC-3( csp)
returns False if an inconsistency is found, True otherwise
inputs: csp, a binary CSP with components (X, D, C)
local variables: queue, a queue of arcs, initially all the arcs in csp
while queue is not empty do
    (X_i, X_j) = REMOVE-FIRST(queue)
    if REVISE(csp, X_i, X_j) then
        if size of D_i = 0 then return False
        for each X_k in X_i.NEIGHBORS − {X_j} do
            add (X_k, X_i) to queue
return true
```

```
function REVISE( csp, X_i, X_j)
returns True iff we revise the domain of X_i
revised = False
for each x in D_i do
    if no value y in D_j allows (x, y) to satisfy the constraint between X_i and X_j then
        delete x from D_i
        revised = True
return revised
```

Complexity of AC-3

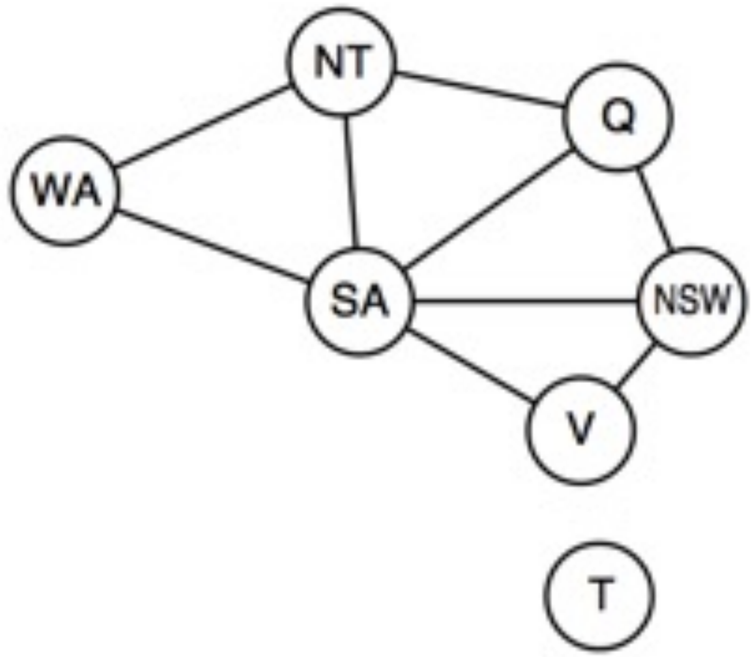
- Let n be the number of variables, and d be the domain size.
- If every node (variable) is connected to the rest of the variables, then we have $n * (n - 1)$ arcs (constraints) $\rightarrow O(n^2)$
- Each arc can be inserted in the queue d times $\rightarrow O(d)$
- Checking the consistency of an arc costs $\rightarrow O(d^2)$.
- Overall complexity is $O(n^2 d^3)$.

Backtracking w/ inference

```
function BACKTRACKING_SEARCH(csp) returns a solution, or failure
    return BACKTRACK({}, csp)

function BACKTRACK(assignment, csp) returns a solution, or failure
    if assignment is complete then return assignment
    var = SELECT_UNASSIGNED-VARIABLE(csp)
    for each value in ORDER_DOMAIN_VALUES (var, assignment, csp)
        if value is consistent with assignment then
            add {var = value} to assignment
            inferences = INFERENCE(csp, var, value)
            if inferences ≠ failure then
                add inferences to assignment
                result = BACKTRACK(assignment, csp)
                if result ≠ failure then return result
        remove {var = value} and inferences from assignment
    return failure
```

Problem structure



- Idea: Leverage the problem structure to make the search more efficient.
- Example: Tasmania is an independent problem.
- Identify the connected component of a graph constraint.
- Work on independent subproblems.

Problem structure

Complexity:

- Let d be the size of the domain and n be the number of variables.
- Time complexity for BTS is $O(d^n)$.
- Suppose we decompose into subproblems, with c variables per subproblem.
- Then we have $\frac{n}{c}$ subproblems.
- c variables per subproblem takes $O(d^c)$.
- The total for all subproblems takes $O(\frac{n}{c}d^c)$ in the worst case.

Problem structure

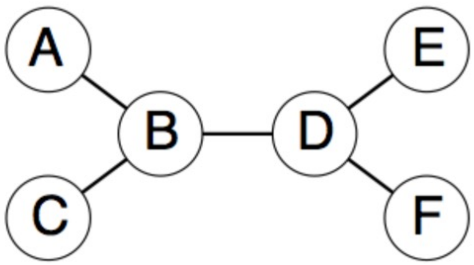
Example:

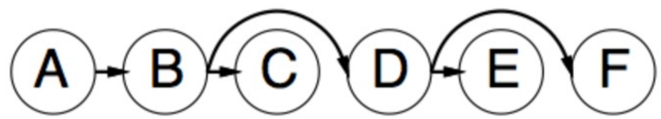
- Assume $n = 80,\ d = 2$.
- Assume we can decompose into 4 subproblems with $c = 20$.
- Assume processing at 10 million nodes per second.
- Without decomposition of the problem we need:
  $2^{80} = 1.2 \times 10^{24}$
  3.83 million years!
- With decomposition of the problem we need:
  $4 \times 2^{20} = 4.2 \times 10^{6}$
  0.4 seconds!

Problem structure

- Turning a problem into independent subproblems is not always possible.
- Can we leverage other graph structures?
- Yes, if the graph is tree-structured or nearly tree-structured.
- A graph is a tree if any two variables are connected by only one path.
- Idea: use DAC, Directed Arc Consistency
- A CSP is said to be directed arc-consistent under an ordering $X_1, X_2, \ldots, X_n$ IFF every X_i is arc-consistent with each X_j for $j > i$.

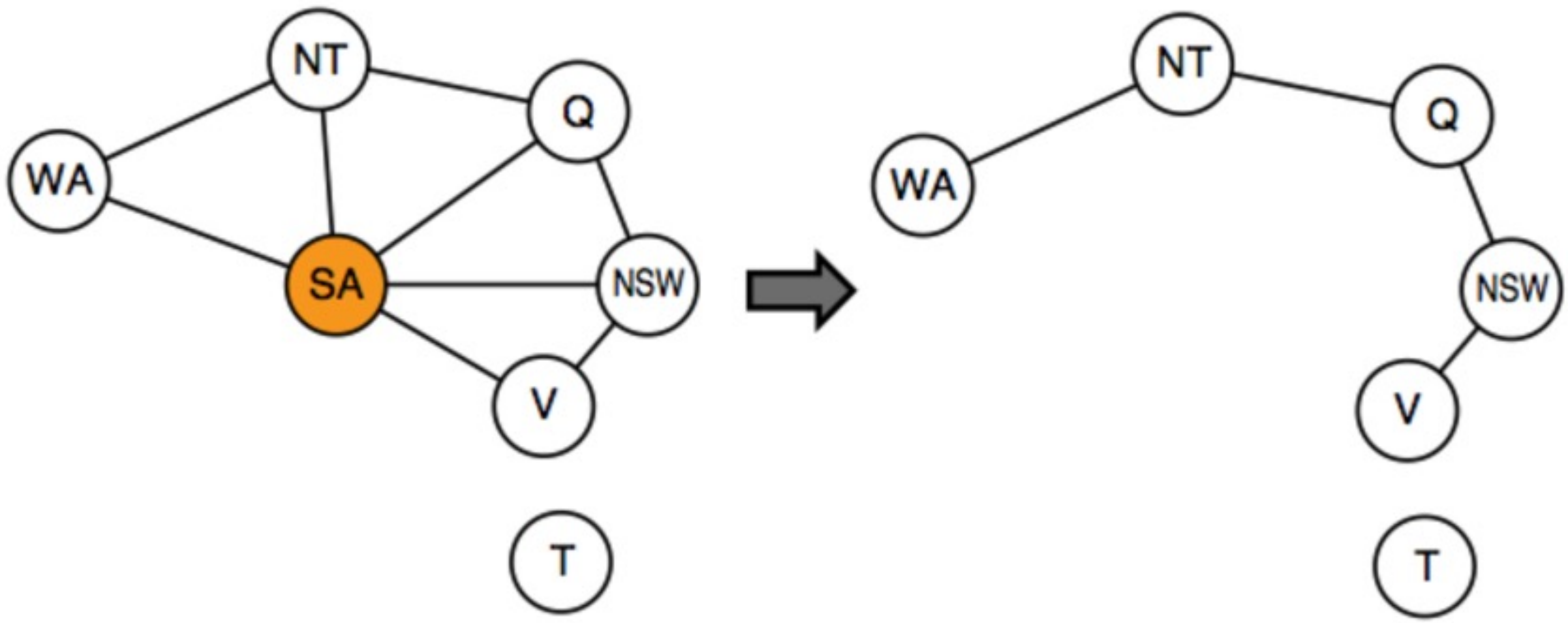
Problem structure





- First pick a variable to the be the root.
- Do a topological sorting: choose an ordering of the variables s.t. each variable appears after its parent in the tree.
- For n nodes, we have $n - 1$ edges.
- Make the tree directed arc-consistent takes $O(n)$
- Each consistency check takes up to $O(d^2)$ (compare d possible values for 2 variables).
- The CSP can be solved in $O(nd^2)$

Nearly tree-structured CSPs



- Assign a variable or a set of variables and prune all the neighbors domains.
- This will turn the constraint graph into a tree :)
- There are other tricks to explore, have fun!

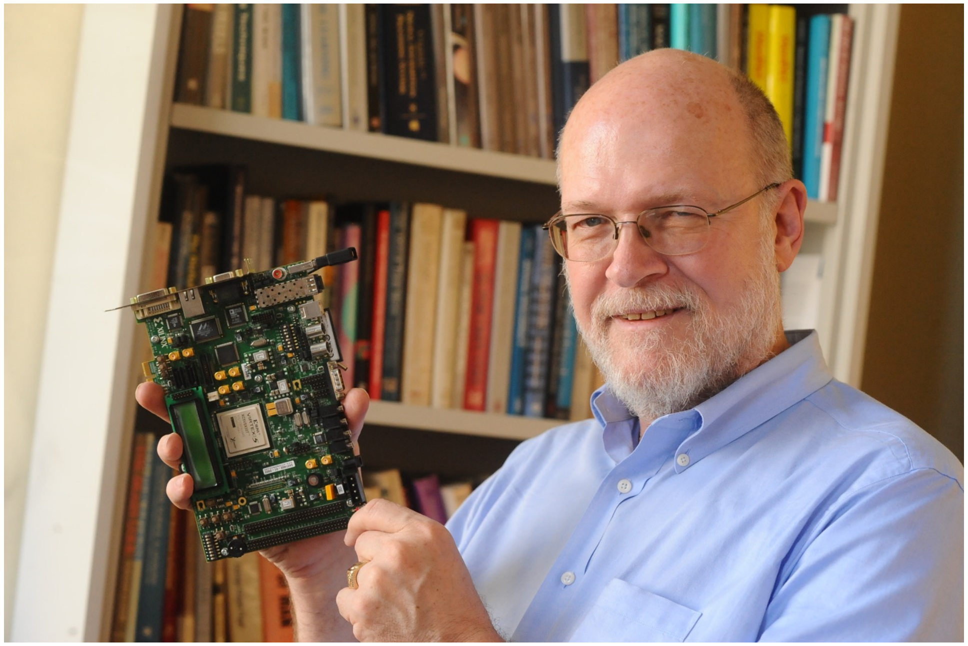
Summary

- CSPs are a special kind of search problems:
  - states defined by values of a fixed set of variables
  - goal test defined by constraints on variable values
- Backtracking = depth-first search with one variable assigned per node
- Variable ordering and value selection heuristics help
- Forward checking prevents assignments that guarantee later failure

Summary

- Constraint propagation (e.g., arc consistency) is an important mechanism in CSPs.
- It does additional work to constrain values and detect inconsistencies.
- Tree-structured CSPs can be solved in linear time
- Further exploration: How can local search be used for CSPs?
- **The power of CSPs: domain-independent, that is you only need to define the problem and then use a solver that implements CSPs mechanisms.**
- Play with CSP solver? Try `http://aispace.org/constraint/`.

David L. Waltz



David L. Waltz
28 May 1943 – 22 March 2012

CCLS founder and leader 2003-2012

David L. Waltz was a computer scientist who made significant contributions in several areas of artificial intelligence, including constraint satisfaction, case-based reasoning and the application of massively parallel computation to AI problems.

Credit

- Artificial Intelligence, A Modern Approach. Stuart Russell and Peter Norvig. Third Edition. Pearson Education.

http://aima.cs.berkeley.edu/