

# Constraint Satisfaction Problems

CE417: Introduction to Artificial Intelligence  
Sharif University of Technology  
Spring 2016

Soleymani

“Artificial Intelligence: A Modern Approach”, 3<sup>rd</sup> Edition, Chapter 6

# Outline

---

- ▶ Constraint Satisfaction Problems (CSP)
  - ▶ Representation for wide variety of problems
  - ▶ CSP solvers can be faster than general state-space searchers
- ▶ Inference in CSPs as a preprocessing stage (AC3 algorithm)
- ▶ Backtracking search for CSPs
  - ▶ Inference during search and heuristics to speedup the backtrack search
- ▶ Problem Structure
- ▶ Local search for CSPs

# What is CSPs?

---

- ▶ In CSPs, the problem is to search for a set of values for the variables (features) so that the assigned values satisfy constraints.
- ▶ CSPs yield a natural representation for a **wide variety of problems**
  - ▶ CSP search algorithms use general-purpose heuristics based on the structure of states

# What is CSPs?

---

## ► Components of a CSP

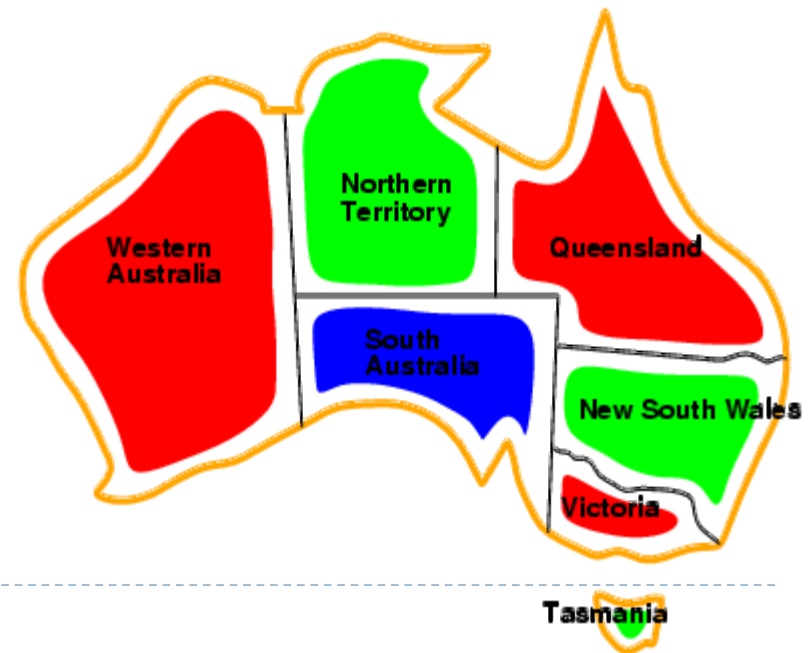
- $X$  is a set of **variables**  $\{X_1, X_2, \dots, X_n\}$
- $D$  is the set of **domains**  $\{D_1, D_2, \dots, D_n\}$  where  $D_i$  is the domain of  $X_i$
- $C$  is a set of **constraints**  $\{C_1, C_2, \dots, C_m\}$ 
  - Each constraint limits the values that variables can take (e.g.,  $X_1 \neq X_2$ )

## ► Solving a CSP

- **State**: An assignment of values to some or all of the variables
- **Solution (goal)**: A complete and consistent assignment
  - Consistent: An assignment that does not violate any constraint
  - Complete: All variables are assigned.

# CSP: Map coloring example

- ▶ Coloring regions with tree colors such that no neighboring regions have the same color
  - ▶ Variables corresponding to regions:  $X = \{WA, NT, Q, NSW, V, SA, T\}$
  - ▶ The domain of all variables is  $\{red, green, blue\}$
  - ▶ Constraints:  $C = \{SA \neq WA, SA \neq NT, SA \neq Q, SA \neq NSW, S \neq V, WA \neq NT, NT \neq Q, Q \neq NSW, NSW \neq V\}$
- ▶ A solution:  
 $\{WA = red, NT = green, Q = red, NSW = green, V = red, T = green\}$



# Example: N-Queens

---

- ▶ Variables:  $\{Q_1, Q_2, \dots, Q_N\}$
- ▶ Domains:  $\{1, 2, \dots, N\}$
- ▶ Constraints:
  - ▶ Implicit:  $\forall i, j \neq i \text{ } non\_threatening(Q_i, Q_j)$
  - ▶ Explicit:  $(Q_i, Q_j) \in \{(1,3), (1,4), \dots, (8,6)\}$

# Boolean satisfiability example

---

- ▶ Given a Boolean expression, is it satisfiable?
  - ▶ *e.g.*,  $((p_1 \wedge p_2) \rightarrow p_3) \vee (\neg p_1 \wedge p_3)$
- ▶ Representing Boolean expression in 3-CNF
- ▶ 3-SAT: find a satisfying truth assignment for 3-CNF
  - ▶ Variables:  $p_1, p_2, \dots, p_n$
  - ▶ Domains:  $\{true, false\}$
  - ▶ Constraints: *all clauses must be satisfied*

# Some real world examples of CSPs

---

- ▶ Assignment problems
  - ▶ e.g., who teaches what class?
- ▶ Timetabling problems
  - ▶ e.g., which class is offered when and where?
- ▶ Transportation scheduling
- ▶ Factory scheduling
  - ▶ Job Shop Scheduling



# Types of variables in CSP formulation

---

## ▶ Discrete variables

### ▶ Finite domains

- ▶  $n$  variables and  $|D_i| = d$  ( $i = 1, \dots, n$ )  $\Rightarrow O(d^n)$  complete assignments

### ▶ Infinite domains

- ▶ e.g., job scheduling, variables are start/end days for each job
  - A constraint language is required (e.g.,  $StartJob_1 + d_1 \leq StartJob_2$ )

## ▶ Continuous variables

- ▶ e.g., exact start/end times for Hubble Space Telescope observations

# Types of constraints in CSP formulation

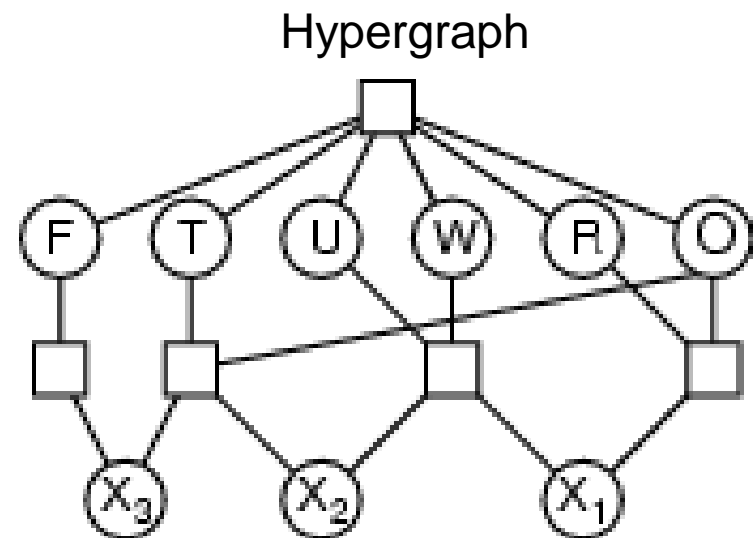
---

- ▶ **Unary** constraints involve a single variable
  - ▶ e.g.,  $SA \neq green$
- ▶ **Binary** constraints involve pairs of variables
  - ▶ e.g.,  $SA \neq WA$
- ▶ **Higher-order** constraints involve 3 or more variables
  - ▶ e.g., cryptarithmic column constraints
  - ▶ Every higher-order finite constraint can be broken into binary constraints, given enough auxiliary constraints

# Cryptarithmic example

- ▶ **Variables:**  $X = \{F, T, U, W, R, O, X_1, X_2, X_3\}$
- ▶ **Domains:**  $\{0, 1, \dots, 9\}$
- ▶ **Constraints:**
  - ▶  $alldiff(F, T, U, W, R, O) \rightarrow$  Global constraint
  - ▶  $O + O = R + 10 \times X_1$
  - ▶  $X_1 + W + W = U + 10 \times X_2$
  - ▶  $X_2 + T + T = O + 10 \times X_3$
  - ▶  $X_3 = F, T \neq 0, F \neq 0$

$$\begin{array}{r}
 \phantom{+} \quad T \quad W \quad O \\
 + \quad T \quad W \quad O \\
 \hline
 F \quad O \quad U \quad R
 \end{array}$$



# Sudoku example

- ▶ Variables: Each (open) square
- ▶ Domains:  $\{1, 2, \dots, 9\}$
- ▶ Constraints:
  - ▶ 9-way *alldiff* for each row
  - ▶ 9-way *alldiff* for each column
  - ▶ 9-way *alldiff* for each region

	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		

(a)

	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

(b)

# Solving CSPs as a standard search problem

## Incremental

---

- ▶ **Initial State:** { }
- ▶ **Actions:** assign a value to an unassigned variable that does not conflict with current assignment
- ▶ **Goal test:** Consistent & complete assignment
- ▶ **Path cost:** not important

# CSP: state space

---

- ▶ Standard search problem:
  - ▶ State is a “black box” with no internal structure (it is a goal or not a goal)
- ▶ Solving CSPs more efficiently by factored representation of states
  - ▶ State is specified by **variables** or features  $X_i$  ( $i = 1, \dots, n$ )
  - ▶ Goal test: Whether each variable has a value that satisfies all the **constraints** on the variable?
- ▶ CSP search algorithms use general-purpose heuristics based on the structure of states
  - ▶ Eliminating large portions of the search space by identifying variable/value combinations that violate the constraints

# CSPs solver phases

---

- ▶ Combination of combinatorial search and heuristics to reach reasonable complexity:
  - ▶ **Search**
    - ▶ Select a new variable assignment from several possibilities of assigning values to unassigned variables
    - ▶ Base of the search process is a **backtracking** algorithm
  - ▶ **Inference** in CSPs (constraint propagation)
    - ▶ “looking ahead” in the search at unassigned variables to eliminate some possible part of the future search space.
      - Using the constraints to reduce legal values for variables
    - ▶ Key idea is **local consistency**

# Inference (looking ahead) in solving CSPs

---

- ▶ Inference as a preprocessing stage
  - ▶ **AC-3 (arc consistency) algorithm**
    - ▶ Removes values from domains of variables (and propagates constraints) to provide all constrained pairs of variables arc consistent.
- ▶ Inference intertwined with search
  - ▶ **Forward checking**
    - ▶ When selecting a value for a variable, infers new domain reductions on neighboring unassigned variables
  - ▶ **Maintaining Arc Consistency (MAC) - Constraint propagation**
    - ▶ Forward checking + recursively propagating constraints when changes are made to the domains
  - ▶ ...



# Base of inference for CSPs: Local consistency

---

- ▶ Node consistency (1-consistency)
  - ▶ Unary constraints are satisfied.
- ▶ Arc consistency (2-consistency)
  - ▶ Binary constraints are satisfied.
- ▶ Path consistency (3-consistency)
- ▶ k- consistency
- ▶ Global constraints

# Arc consistency

---

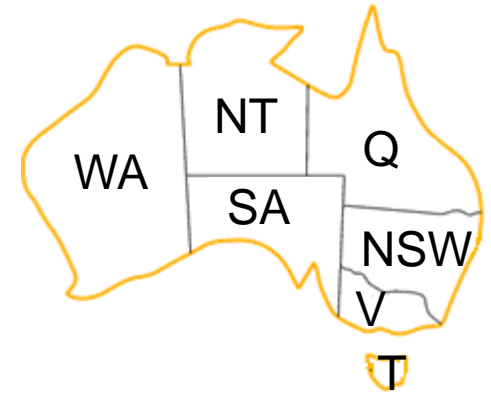
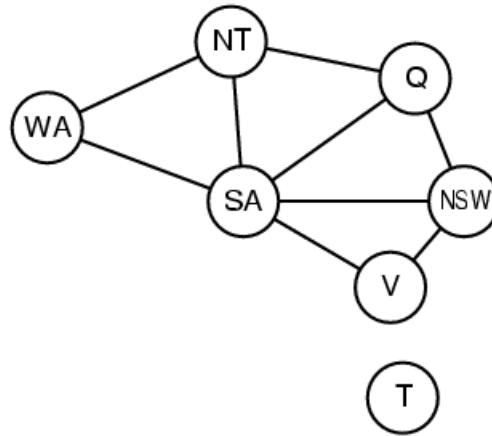
- ▶  $X_i$  is arc-consistent with respect to  $X_j$   
if for every value in  $D_i$  there is a consistent value in  $D_j$

- ▶ **Example**

- ▶ Variables:  $X = \{X_1, X_2\}$
- ▶ Domain:  $\{0, 1, 2, \dots, 9\}$
- ▶ Constraint:  $X_1 = X_2^2$
  
- ▶ Is  $X_1$  arc-consistent w.r.t.  $X_2$ ?
  - ▶ No, to be arc-consistent  $\text{Domain}(X_1) = \{0, 1, 4, 9\}$
- ▶ Is  $X_2$  arc-consistent w.r.t.  $X_1$ ?
  - ▶ No, to be arc-consistent  $\text{Domain}(X_2) = \{0, 1, 2, 3\}$

# Constraint graph

- ▶ Nodes are variables, arcs are constraints



- ▶ Enforcing **local consistency** in each part of the graph can cause inconsistent values to be eliminated

# Arc consistency algorithm (AC-3)

**function** *AC\_3*(*csp*) **returns** false if an inconsistency is found and 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**

$(X_i, X_j) \leftarrow \text{REMOVE\_FIRST}(\text{queue})$

**if** *REVISE*(*csp*,  $X_i$ ,  $X_j$ ) **then**

**if** size of  $D_i = 0$  **then return** false

**for each**  $X_k$  **in**  $X_i.\text{NEIGHBORS} - \{X_j\}$  **do**

add  $(X_k, X_i)$  to *queue*

**function** *REVISE*(*csp*,  $X_i$ ,  $X_j$ ) **returns** true iff we revise the domain of  $X_i$

*revised*  $\leftarrow$  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*  $\leftarrow$  true

**return** *revised*



# AC-3


---

For each arc  $(X_i, X_j)$  in the queue

Remove it from queue

Makes  $X_i$  arc-consistent with respect to  $X_j$

- 1) If  $D_i$  remains unchanged then continue
- 2) If  $|D_i| = 0$  then return false
- 3) For each neighbor  $X_k$  of  $X_i$  except to  $X_j$  do  
add  $(X_k, X_i)$  to queue



If domain of  $X_i$  loses a value,  
neighbors of  $X_i$  must be rechecked

- ▶ Removing a value from a domain may cause further inconsistency, so we have to repeat the procedure until everything is consistent.
- ▶ When queue is empty, resulted CSP is equivalent to the original CSP.
  - ▶ Same solution (usually reduced domains speed up the search)

## AC-3: time complexity

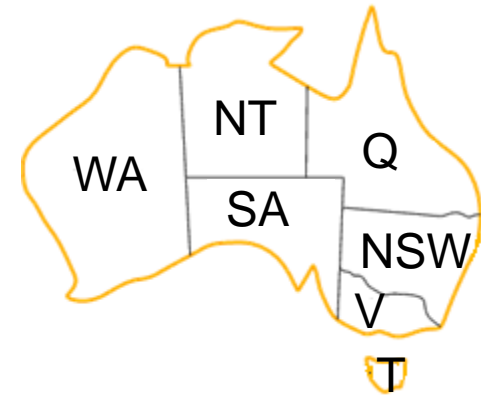
---

- ▶ Time complexity ( $n$  variables,  $c$  binary constraints,  $d$  domain size):  $O(cd^3)$ 
  - ▶ Each arc  $(X_k, X_i)$  is inserted in the queue at most  $d$  times.
    - ▶ At most all values in domain  $X_i$  can be deleted.
  - ▶ Checking consistency of an arc:  $O(d^2)$

# Arc consistency: map coloring example

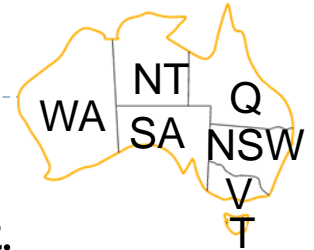
---

- ▶ For general map coloring problem all pairs of variables are arc-consistent if  $|D_i| \geq 2 (i = 1, \dots, n)$
- ▶ Arc consistency can do nothing.
  - ▶ Fails to make enough inference
- ▶ We need stronger notion of consistency to detect failure at start.
  - ▶ 3-consistency (path consistency): for any consistent assignment to each set of two variables, a consistent value can be assigned to any other variable.
  - ▶ When we have two colors, both of the possible assignments to the set  $\{WA, SA\}$  are inconsistent with  $NT$ .



# k-consistency

---



- ▶ Arc consistency does not detect all inconsistencies
  - ▶ Partial assignment  $\{WA = red, NSW = blue\}$  is inconsistent.
- ▶ A CSP is  $k$ -consistent if for any set of  $k - 1$  variables and for any consistent assignment to those variables, a consistent value can always be assigned to any  $k$ th variable.
  - ▶ E.g. 1-consistency = node-consistency
  - ▶ E.g. 2-consistency = arc-consistency
  - ▶ E.g. 3-consistency = path-consistency



# Which level of consistency?

---

- ▶ **Trade off** between the required time to establish  $k$ -consistency and the amount of the eliminated search space.
  - ▶ If establishing consistency is slow, this can slow the search down to the point where no propagation is better.
- ▶ Establishing  $k$ -consistency needs exponential time and space in  $k$  (in the worst case)
- ▶ Commonly computing **2-consistency** and less commonly 3-consistency

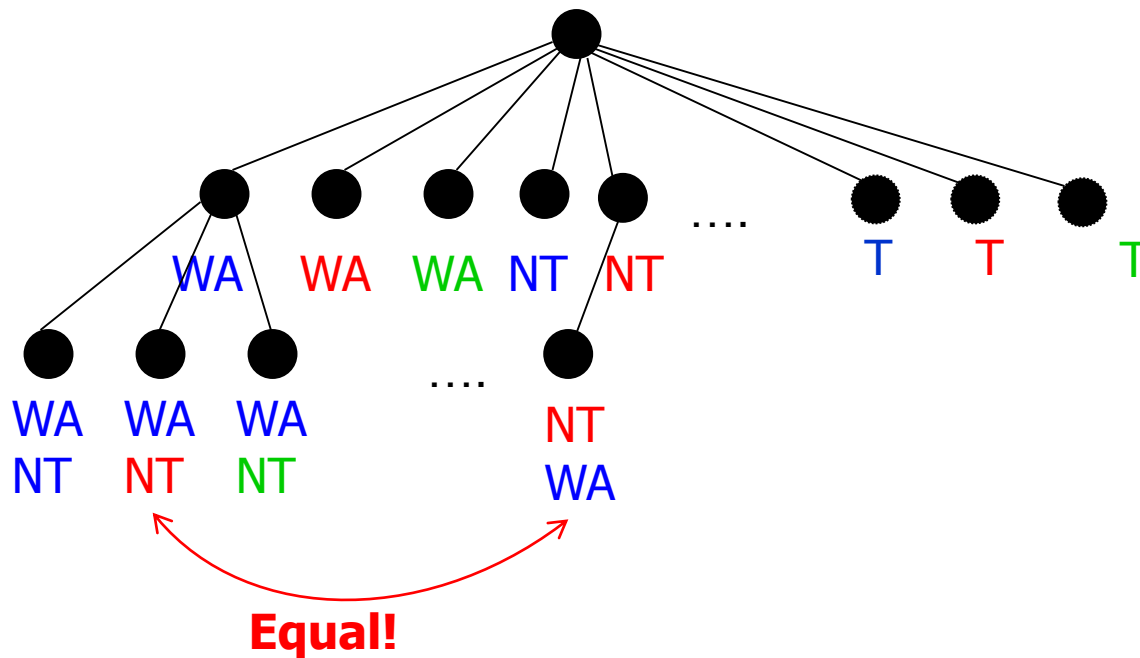
# Properties of CSPs as a standard search problem

---

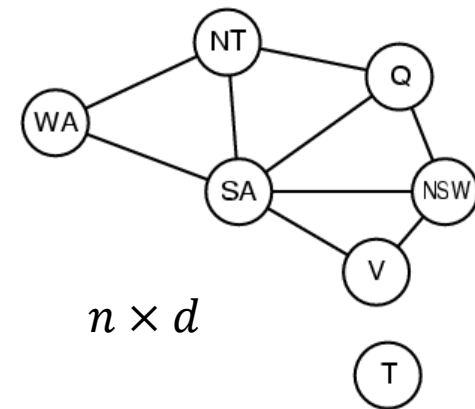
- ▶ Generic problem formulation: same formulation for all CSPs
- ▶ Every solution appears at depth  $n$  with  $n$  variables
- ▶ Branching factor is  $nd$  at the top level,  $b = (n - l)d$  at depth  $l$ , hence there are  $n! d^n$  leaves.
  - ▶ However, there are only  $d^n$  complete assignments.
- ▶ Which search algorithm is proper?
  - ▶ Depth-limited search

# Assignment community

- ▶ When assigning values to variables, we reach the same partial assignment regardless of the order of the variables chosen to assign values to.



There are  $n! \times d^n$  leaves in the tree but only  $d^n$  distinct states!



$n \times d$

$(n \times d) \times ((n - 1) \times d)$

$\vdots$   
 $n! \times d^n$

# Techniques that speed up CSP solvers

---

- ▶ Many intractable problems for regular state-space search, can be solved quickly by formulating as CSPs using these techniques:

- ▶ **Backtracking**

- ▶ when a partial assignment is found inconsistent, discards further refinements of this (not promising) assignment.

- ▶ **Ordering heuristics**

- ▶ order of variables to be tried and selection of values to be assigned.

- ▶ **Domains reduction**

- ▶ eliminates values causing future failure from domains of variables (before or during search).

- ▶ **Graph structure**

- ▶ Decompose the whole problem into disjoint or least connected sub-problems

# Backtracking search

---

- ▶ Depth-first search for CSPs with single-variable assignments is called **backtracking search**
  - ▶ assigns one variable at each level (eventually they all have to be assigned.)
- ▶ Naïve backtracking is not generally efficient for solving CSPs.
  - ▶ More heuristics will be introduced later to speedup it.

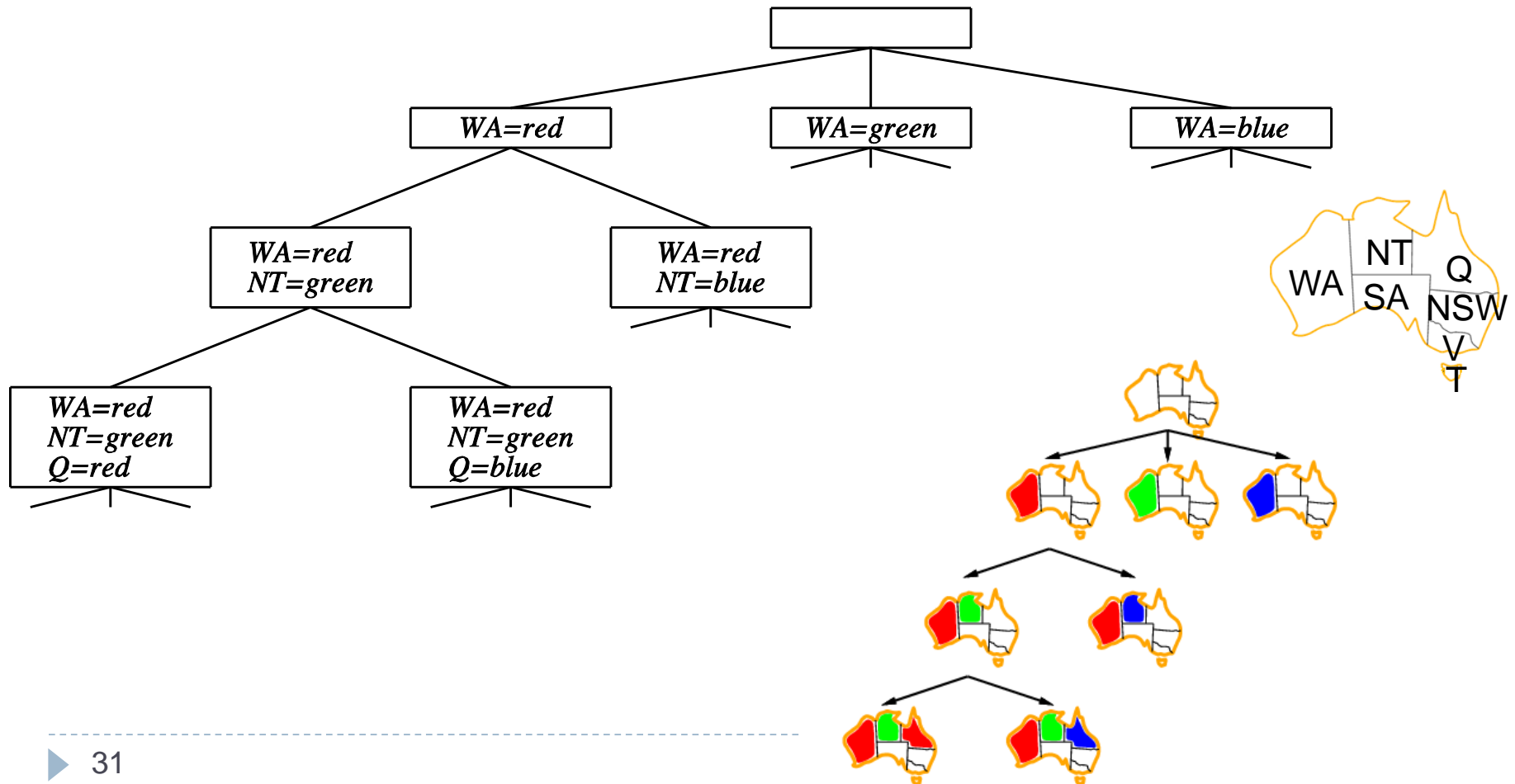
# Backtracking search

---

- ▶ Nodes are partial assignments
- ▶ Incremental completion
  - ▶ Each partial candidate is the parent of all candidates that differ from it by a single extension step.
- ▶ Traverses the search tree in depth first order.
- ▶ At each node  $c$ 
  - ▶ If it cannot be completed to a valid solution, the whole sub-tree rooted at  $c$  is skipped (not promising branches are *pruned*).
  - ▶ Otherwise, the algorithm (1) checks whether  $c$  itself is a valid solution, returns it; and (2) recursively enumerates all sub-trees of  $c$ .

# Search tree

- Variable assignments in the order:  $WA, NT, Q, \dots$



# General backtracking search

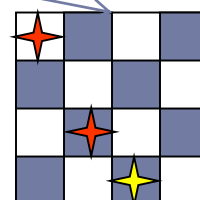
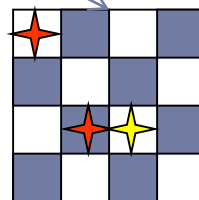
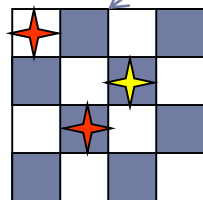
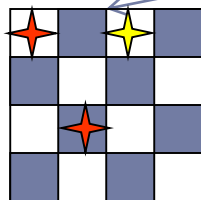
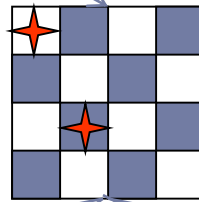
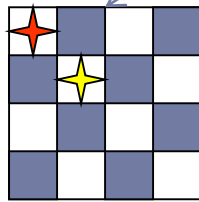
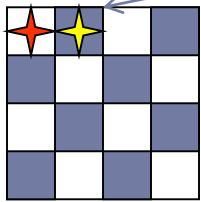
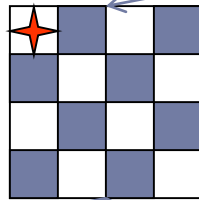
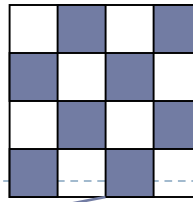
```
function BACKTRACK( $v$ ) returns a solution, or failure  
  if there is a solution at  $v$  then return solution  
  for each child  $u$  of  $v$  do  
    if Promising( $u$ ) then  
       $result \leftarrow BACKTRACK(u)$   
      if  $result \neq \text{failure}$  return  $result$   
  return failure
```



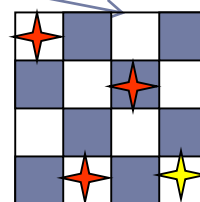
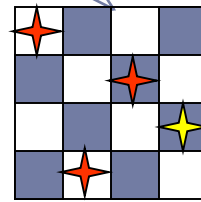
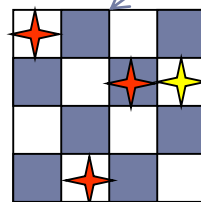
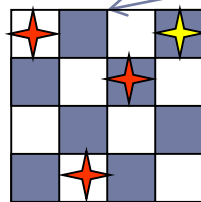
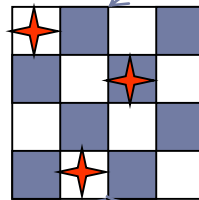
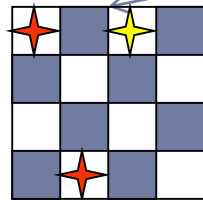
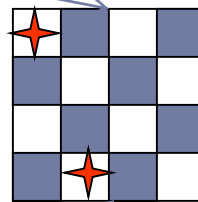
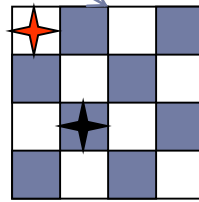
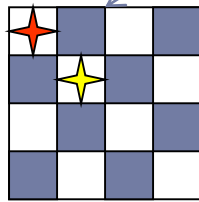
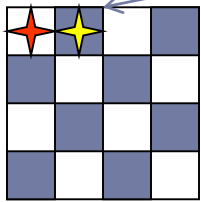
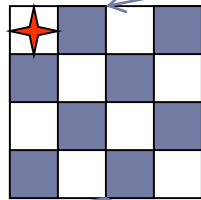
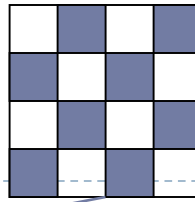
```
function BACKTRACK( $assignment, csp$ ) returns an assignment, or failure  
  if  $assignment$  is complete then return  $assignment$   
   $var \leftarrow$  select an unassigned variable  
  for each  $val$  in Domain( $var$ ) do  
    if Consistent( $assignment \cup \{var \leftarrow value\}, csp$ ) then  
       $result \leftarrow BACKTRACK(assignment \cup \{var \leftarrow value\}, csp)$   
      if  $result \neq \text{failure}$  return  $result$   
  return failure
```



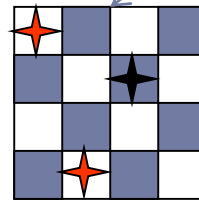
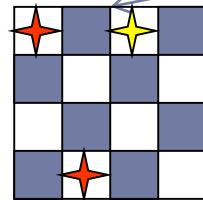
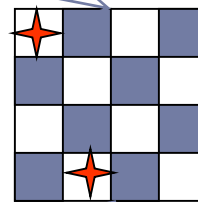
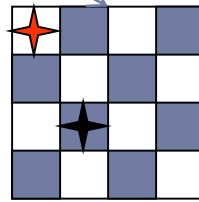
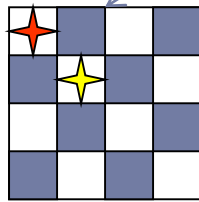
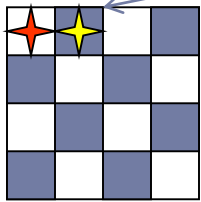
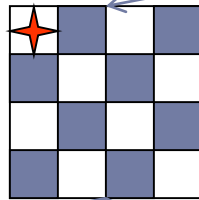
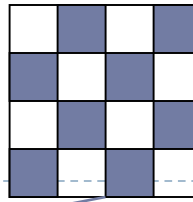
# 4-Queens



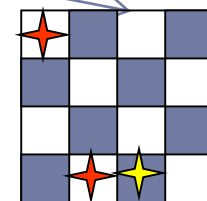
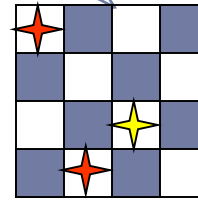
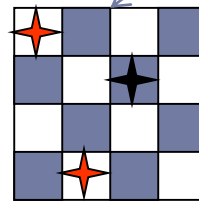
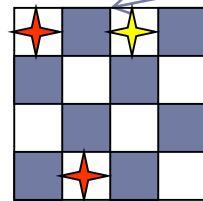
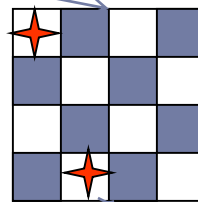
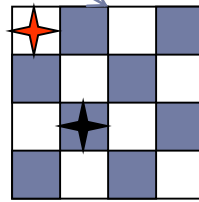
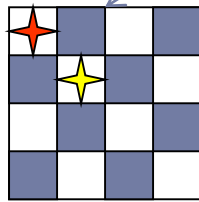
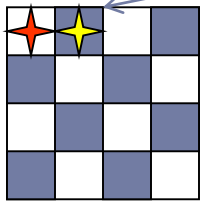
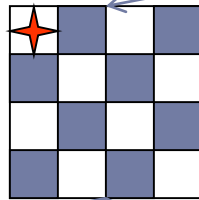
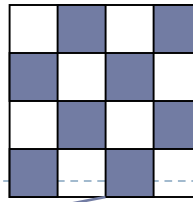
# 4-Queens



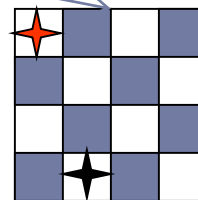
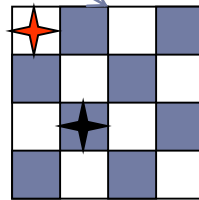
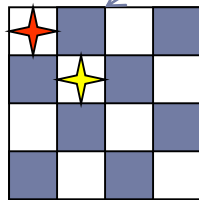
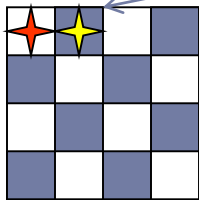
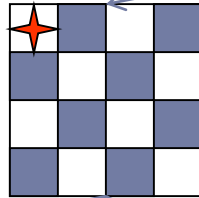
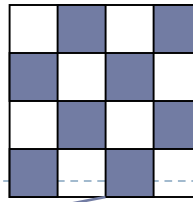
# 4-Queens



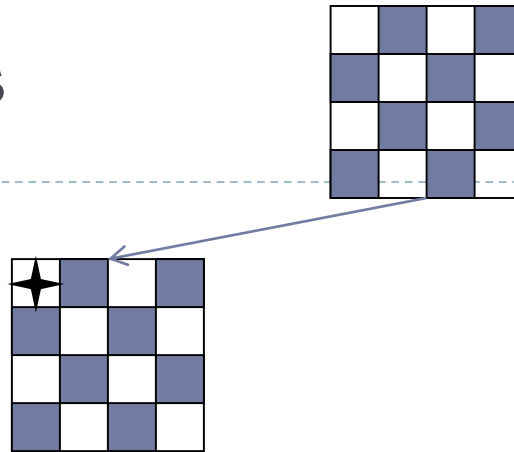
# 4-Queens



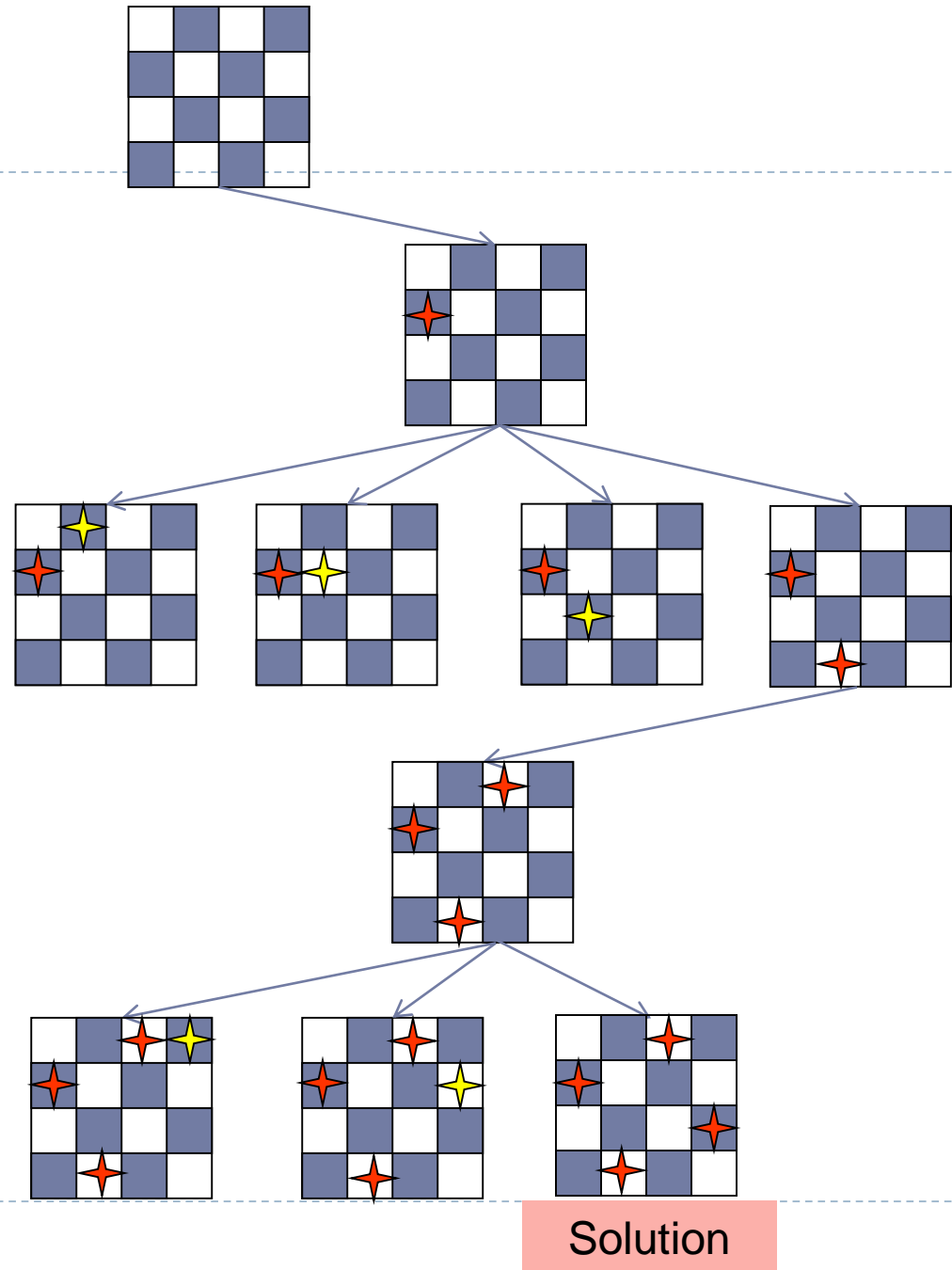
# 4-Queens



# 4-Queens

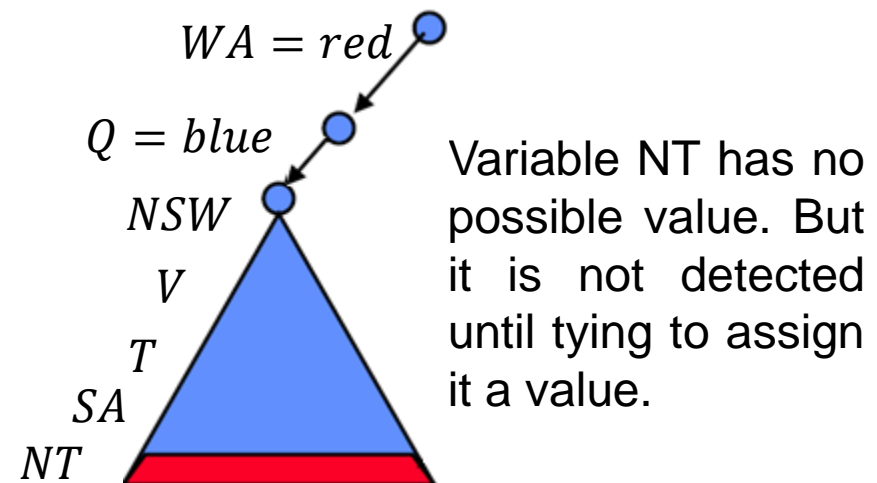
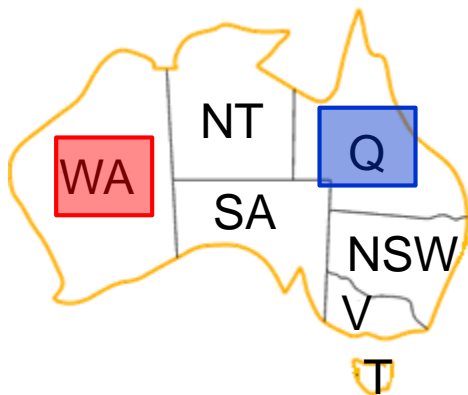


# 4-Queens



# Naïve backtracking (late failure)

- ▶ Map coloring with three colors
  - ▶  $\{WA = red, Q = blue\}$  can not be completed.
  - ▶ However, the backtracking search does not detect this before selecting but  $NT$  and  $SA$  variables





# Inference during the search process

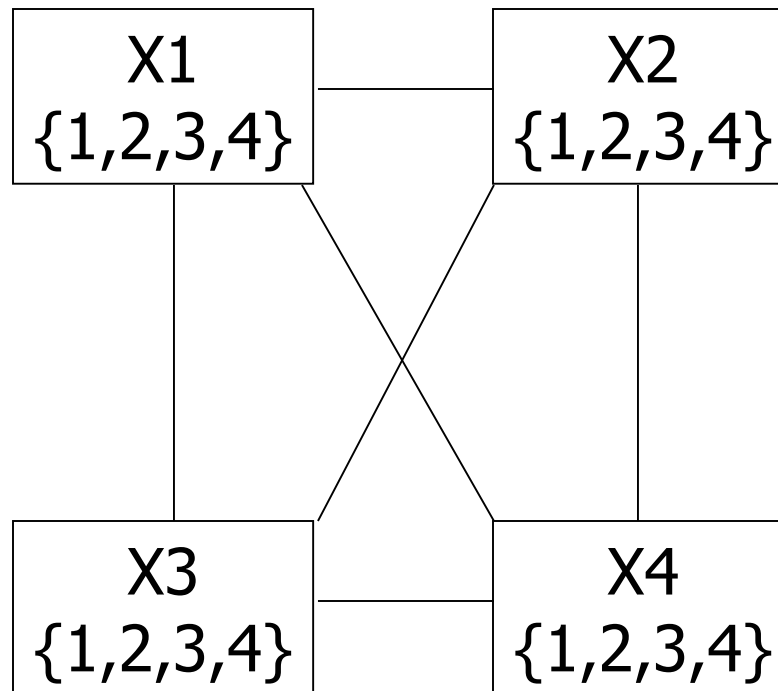
---

- ▶ It can be more powerful than inference in the preprocessing stage.
- ▶ Interleaving search and inference

# Example: 4-Queens

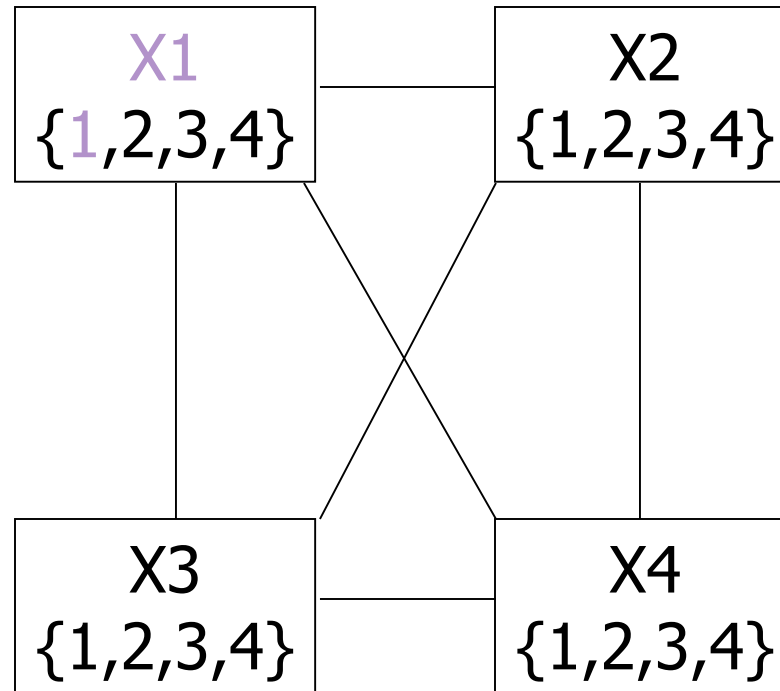
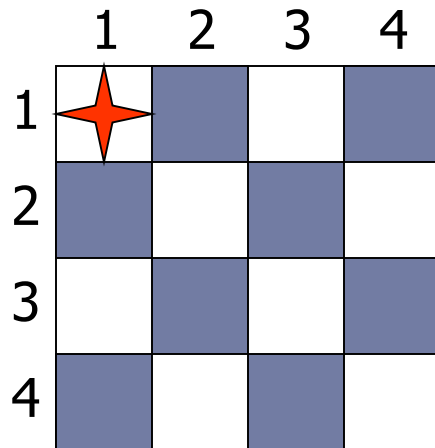
---

	1	2	3	4
1				
2				
3				
4				



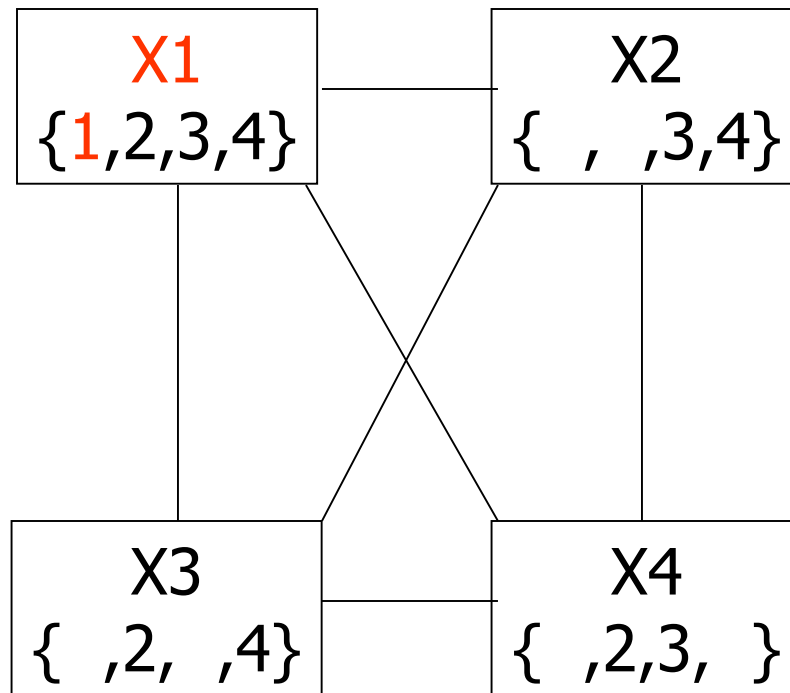
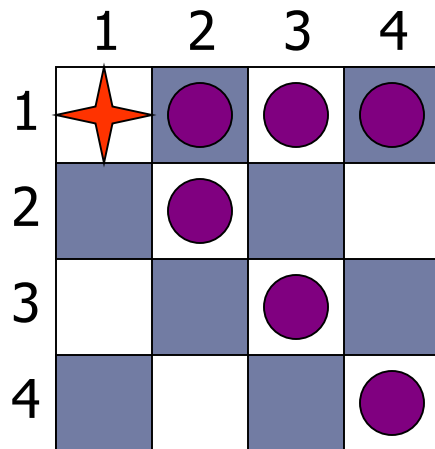
# Example: 4-Queens

---

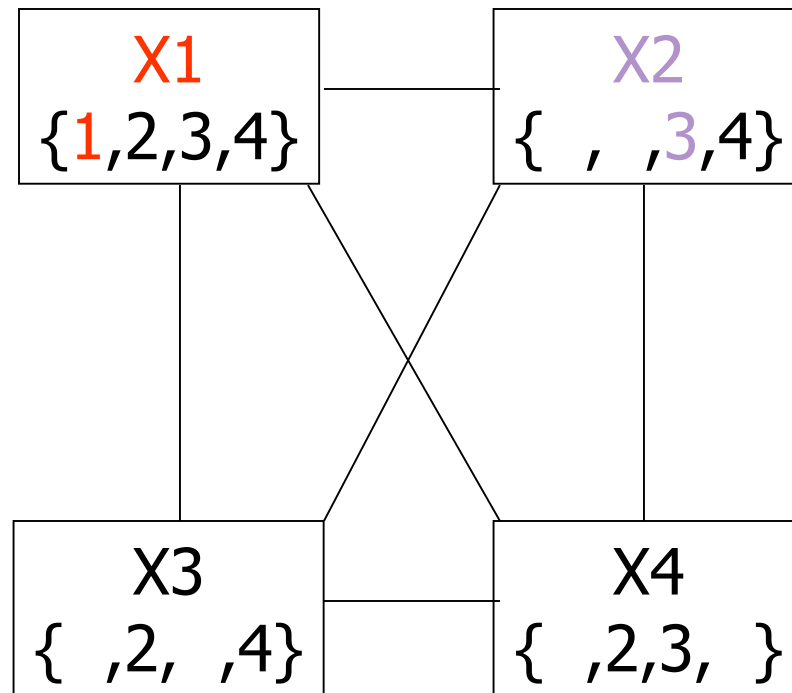
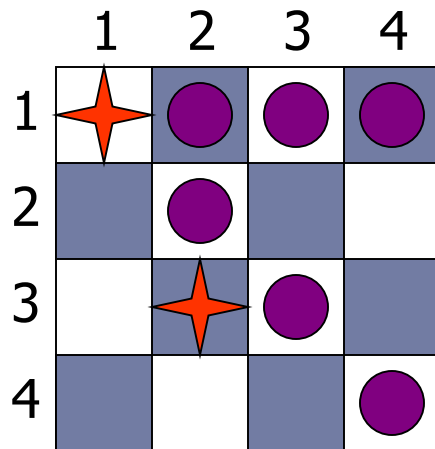


# Example: 4-Queens

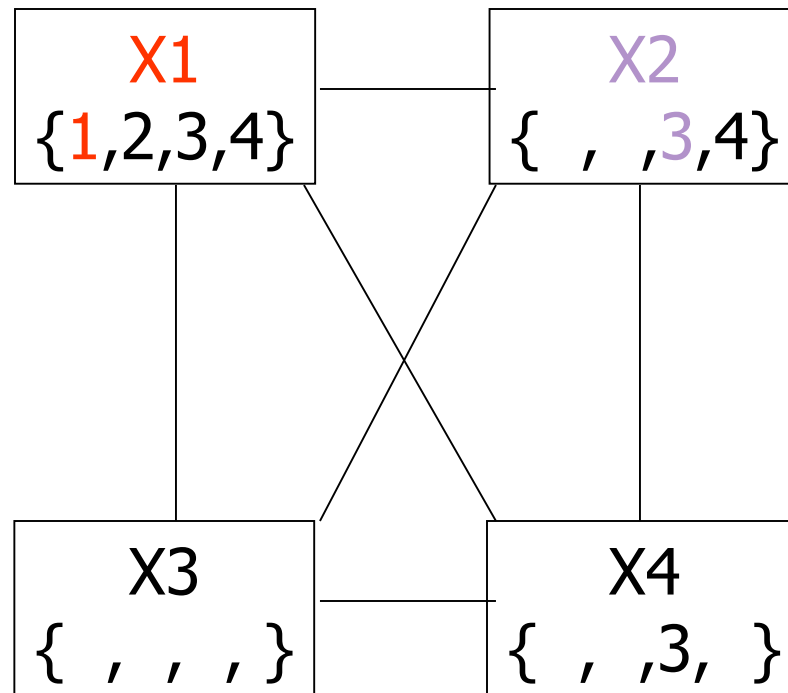
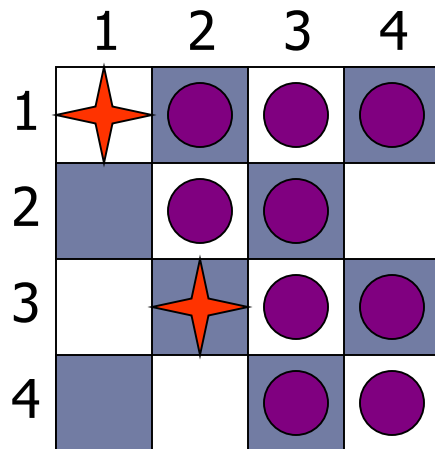
---



# Example: 4-Queens

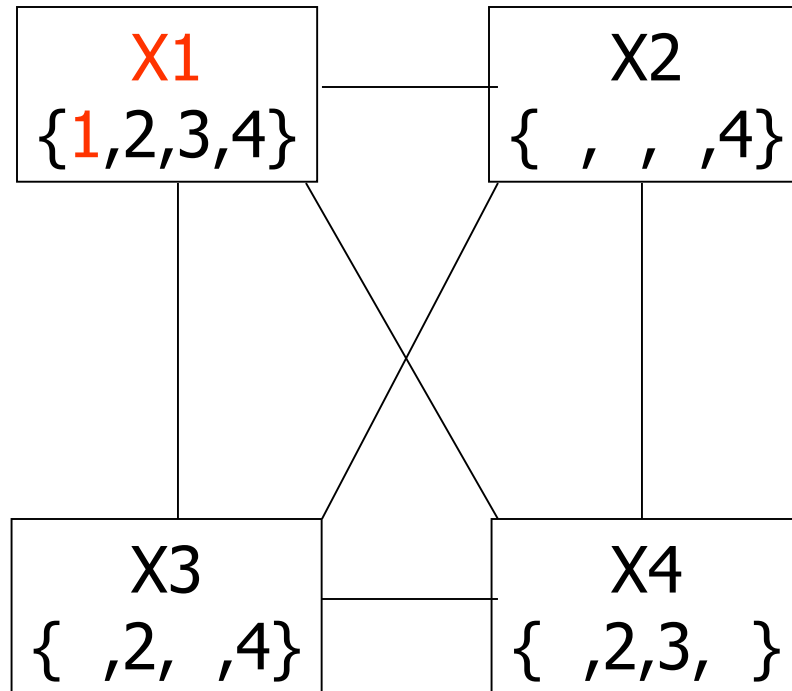
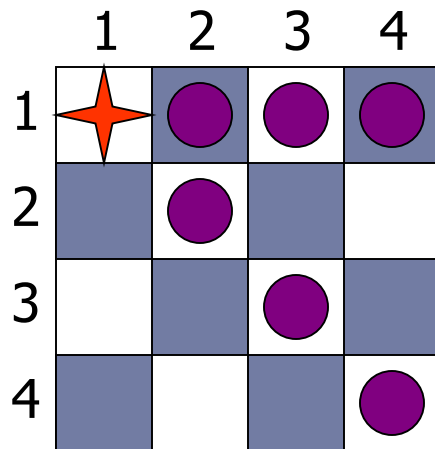


# Example: 4-Queens

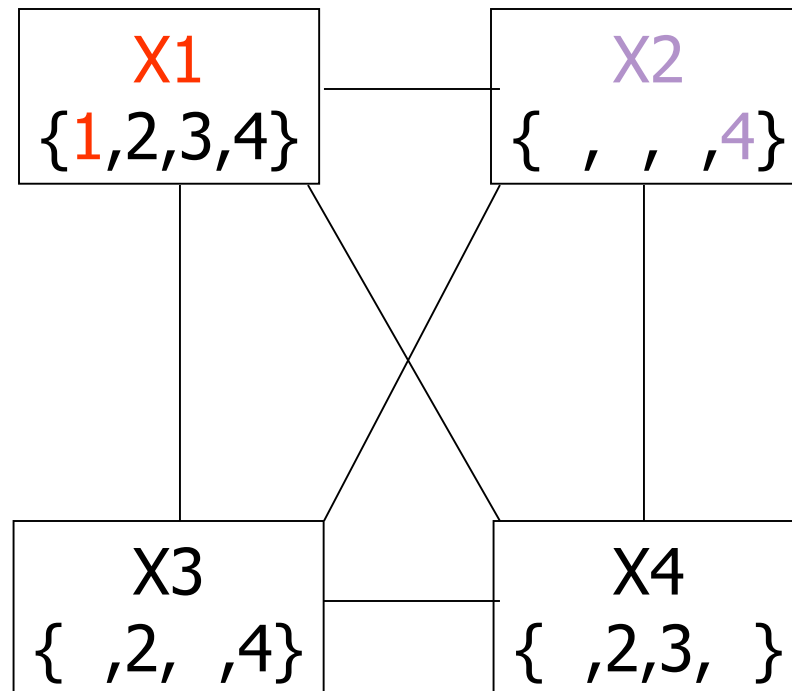
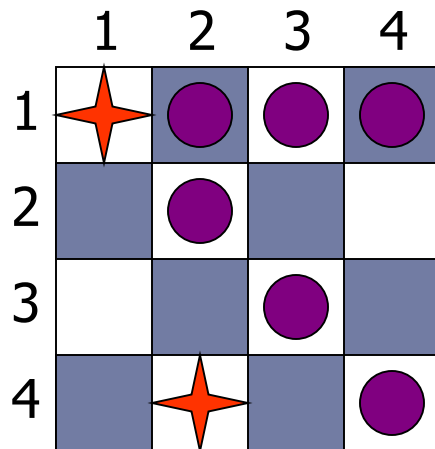


# Example: 4-Queens

---

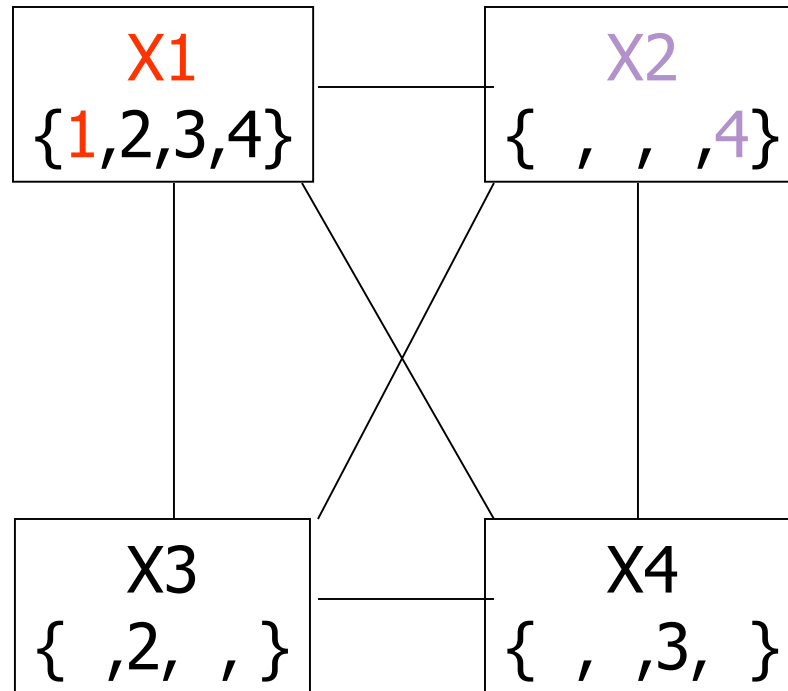
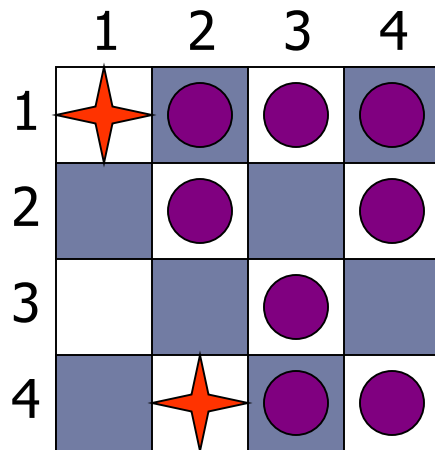


# Example: 4-Queens



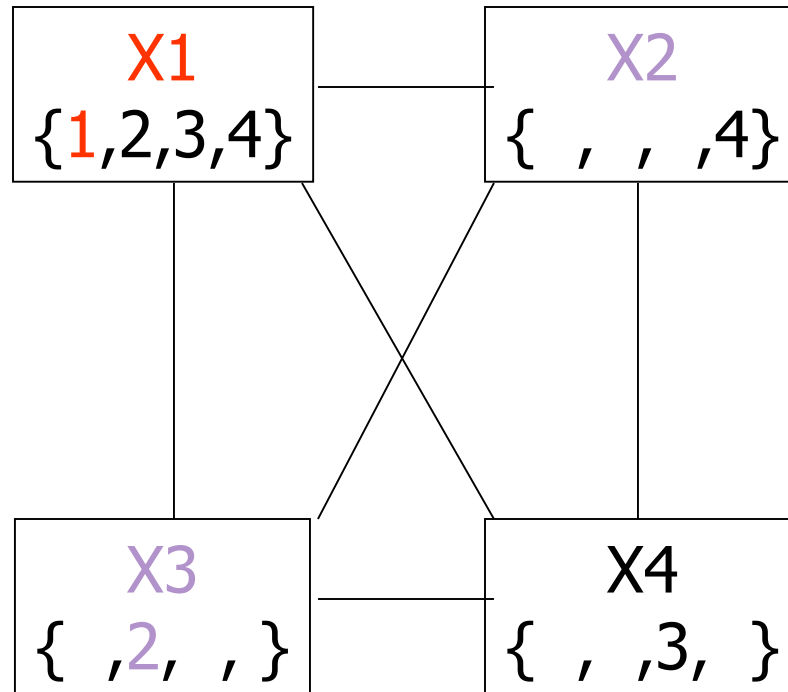
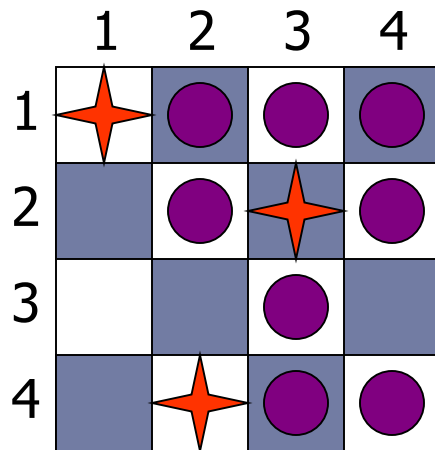


# Example: 4-Queens



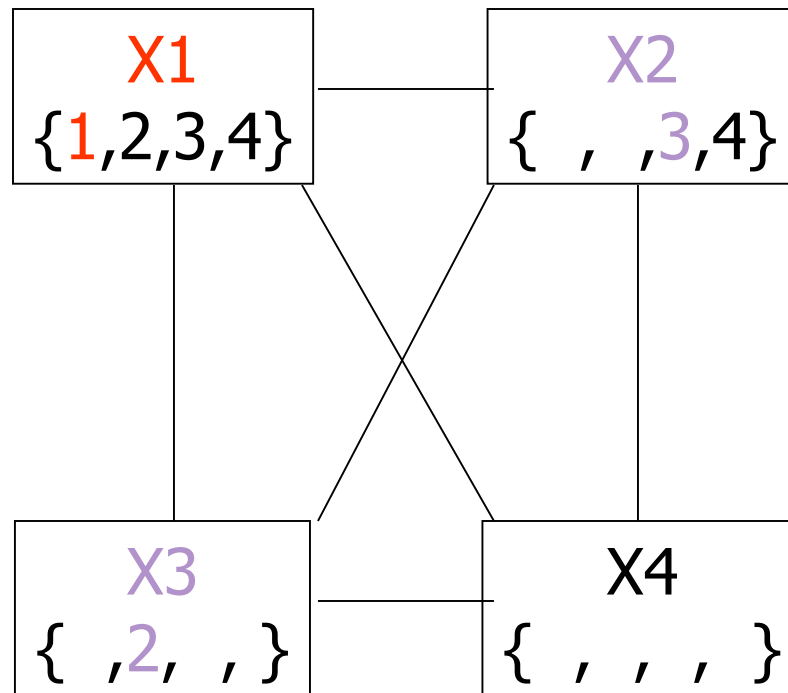
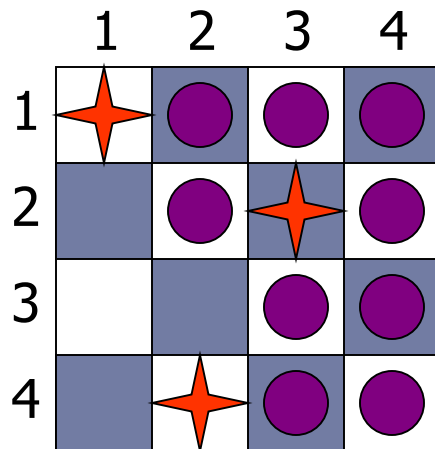
# Example: 4-Queens

---



# Example: 4-Queens

---



We will call this type of inference during search as Forward Checking soon.

# Solving CSP efficiently

---

- ▶ Which variable should be assigned next?
  - ▶ *SELECT\_UNASSIGNED\_VARIABLE*
- ▶ In what order should values of the selected variable be tried?
  - ▶ *ORDER\_DOMAIN\_VALUES*
- ▶ What inferences should be performed at each step in the search?
  - ▶ *INFERENCE*

# CSP backtracking search

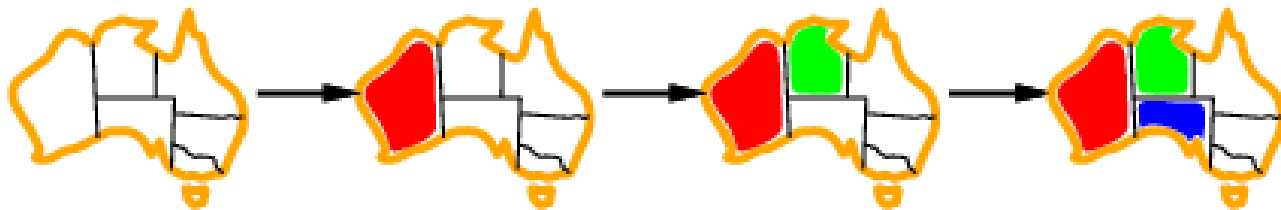
**function** *BACKTRACKIN\_SEARCH*(*csp*) **returns** a solution, or failure  
**return** *BACKTRACK*(*{ }*, *csp*)

**function** *BACKTRACK*(*assignment*, *csp*) **returns** a solution, or failure  
**if** *assignment* is complete **then return** *assignment*  
*var*  $\leftarrow$  *SELECT\_UNASSIGNED\_VARIABLE*(*csp*)  
**for each** *value* **in** *ORDER\_DOMAIN\_VALUES*(*var*, *assignment*, *csp*) **do**  
    **if** *value* is consistent with *assignment* **then**  
        add {*var* = *value*} to *assignment*  
        *inferences*  $\leftarrow$  *INFERENCE*(*csp*, *var*, *value*)  
        **if** *inferences*  $\neq$  failure **then**  
            add *inferences* to *assignment*  
            *result*  $\leftarrow$  *BACKTRACK*(*assignment*, *csp*)  
            **if** *result*  $\neq$  failure **then return** *result*  
        remove {*var* = *value*} and *inferences* from *assignment*  
**return** failure

# Minimum Remaining Values (MRV)

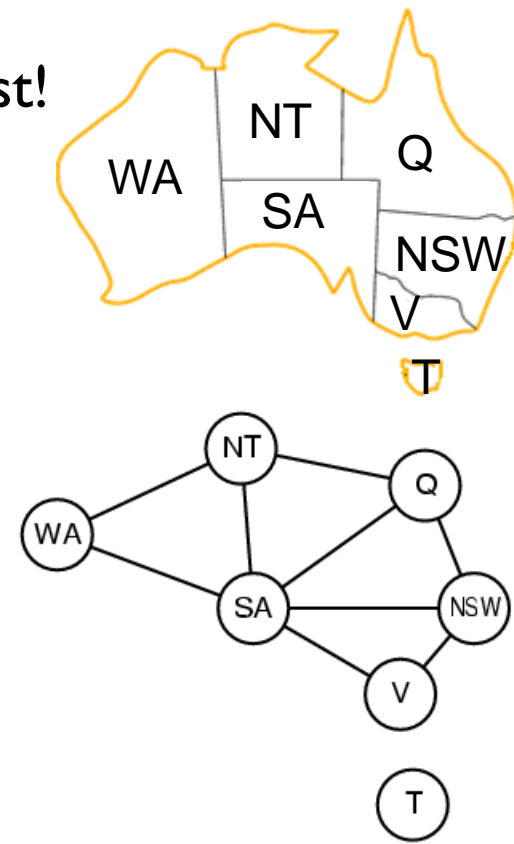
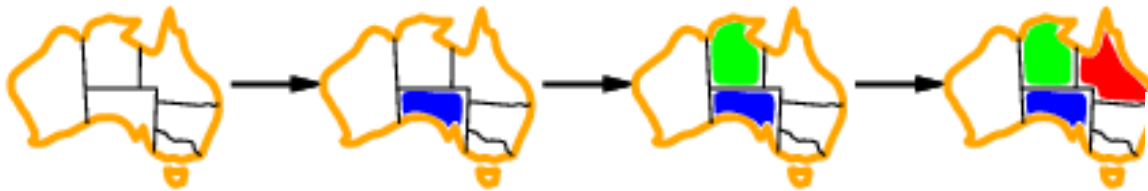
---

- ▶ Chooses the variable with the fewest legal values
  - ▶ Fail first
- ▶ Also known as **Most Constrained Variable (MCS)**
- ▶ Most likely to cause a failure soon and so pruning the search tree



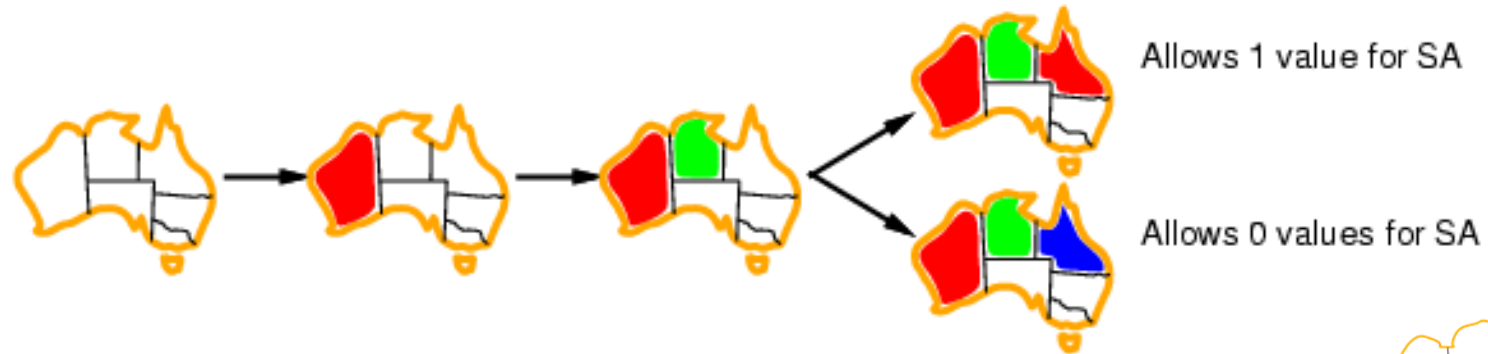
# Degree heuristic

- ▶ Tie-breaker among MRV variables
- ▶ Degree heuristic: choose the variable with the most constraints on remaining variables
  - ▶ To choose one who interferes the others most!
  - ▶ reduction in branching factor

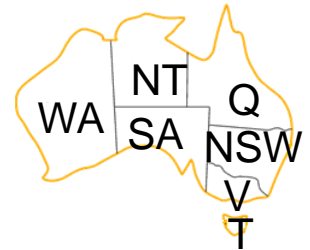


# Least constraining value

- ▶ Given a variable, choose the least constraining value:
  - ▶ one that rules out the fewest values in the remaining variables
  - ▶ leaving maximum flexibility for subsequent variable assignments
    - ▶ Fail last (the most likely values first)



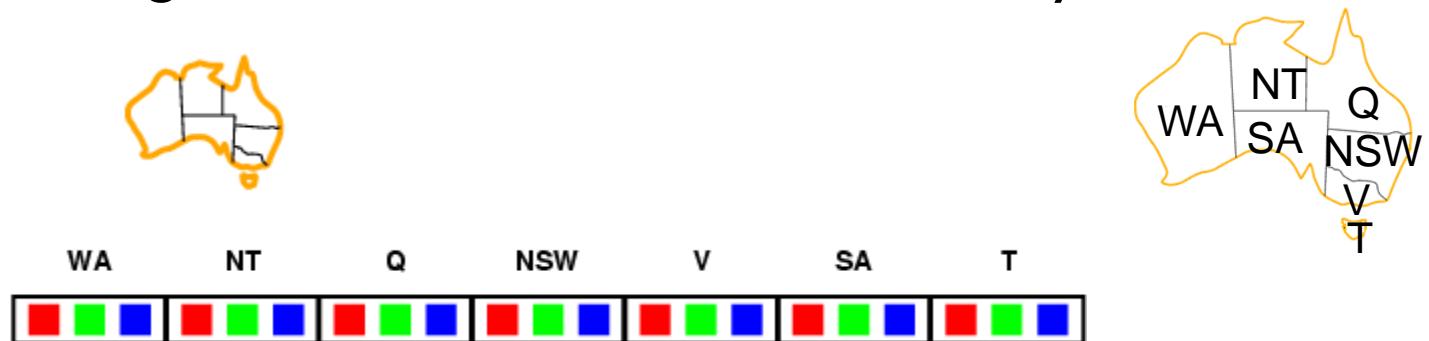
- ▶ Assumption: we only need one solution





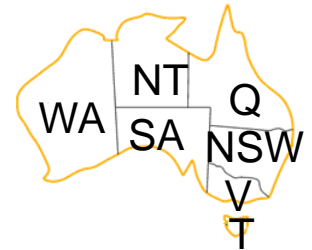
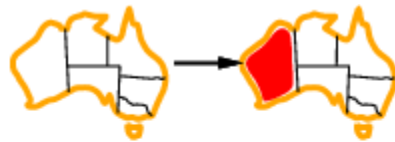
# Forward Checking (FC)

- ▶ When selecting a value for a variable, infer new domain reductions on neighboring unassigned variables.
  - ▶ Terminate search when a variable has no legal value
- ▶ When  $X$  is assigned, FC establishes arc-consistency for it.



# Forward Checking (FC)

- ▶ When selecting a value for a variable, infer new domain reductions on neighboring unassigned variables.
  - ▶ Terminate search when a variable has no legal value
- ▶ When  $X$  is assigned, FC establishes arc-consistency for it.



WA	NT	Q	NSW	V	SA	T
<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>
<div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>

$WA = red$

# Forward Checking (FC)

- ▶ When selecting a value for a variable, infer new domain reductions on neighboring unassigned variables.
  - ▶ Terminate search when a variable has no legal value
- ▶ When  $X$  is assigned, FC establishes arc-consistency for it.



WA	NT	Q	NSW	V	SA	T
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>
<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>	<div><div></div><div></div><div></div></div>

$WA = red$

$Q = green$

# Forward Checking (FC)

- ▶ When selecting a value for a variable, infer new domain reductions on neighboring unassigned variables.
  - ▶ Terminate search when a variable has no legal value
- ▶ When  $X$  is assigned, FC establishes arc-consistency for it.



	WA	NT	Q	NSW	V	SA	T
	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>
$WA = red$	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>
$Q = green$	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>
$V = blue$	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>	<div><div>■</div><div>■</div><div>■</div></div>

$\Rightarrow \{WA = red, Q = green, V = blue\}$  is an inconsistent partial assignment

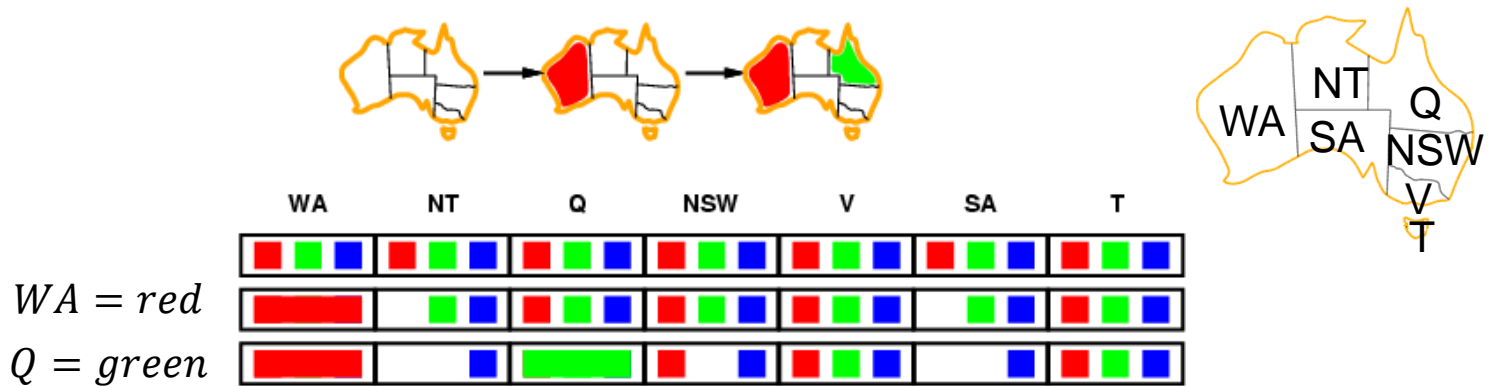
# Forward Checking (FC)

---

- ▶ Is this claim true? “No need to FC if arc consistency (AC-3) have been done as a preprocessing stage”.
- ▶ Combination of MRV heuristic and FC is more effective.
  - ▶ FC incrementally computes the information that MRV needs.

# Constraint propagation

- ▶ FC makes the current variable arc-consistent but does not make all the other variables arc-consistent



- ▶ NT and SA cannot both be blue!
  - ▶ FC does not look far enough ahead to find this inconsistency
- ▶ **Maintaining Arc Consistency (MAC) - Constraint propagation**
  - ▶ Forward checking + recursively propagating constraints when changing domains (similar to AC-3 but only arcs related to the current variable are put in the queue at start)

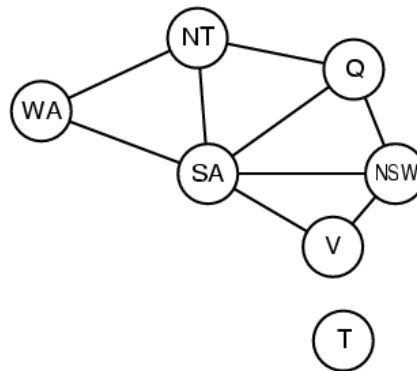
# CSP backtracking search

**function** *BACKTRACKIN\_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*  $\leftarrow$  *SELECT\_UNASSIGNED\_VARIABLE*(*csp*)  
**for each** *value* **in** *ORDER\_DOMAIN\_VALUES*(*var*, *assignment*, *csp*) **do**  
    **if** *value* is consistent with *assignment* **then**  
        add {*var* = *value*} to *assignment*  
        *inferences*  $\leftarrow$  *INFERENCE*(*csp*, *var*, *value*)  
        **if** *inferences*  $\neq$  failure **then**  
            add *inferences* to *assignment*  
            *result*  $\leftarrow$  *BACKTRACK*(*assignment*, *csp*)  
            **if** *result*  $\neq$  failure **then return** *result*  
        remove {*var* = *value*} and *inferences* from *assignment*  
**return** failure

# Graph Structure

- ▶ Connected components as **independent sub-problems**
  - ▶ The color of  $T$  is independent of those of other regions



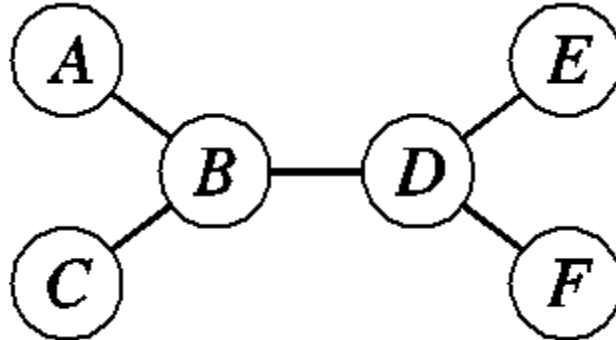
- ▶ Suppose each sub-problem has  $h$  variables out of  $n$ 
  - ▶ Worst-case solution cost is  $O((n/h)(d^h))$  that is linear in  $n$
- ▶ Example:  $n = 80$ ,  $d = 2$ ,  $h = 20$  (processing:  $10^6$  nodes/sec)
  - ▶ 40 billion years
  - ▶ 4 seconds



# Tree structured CSPs

---

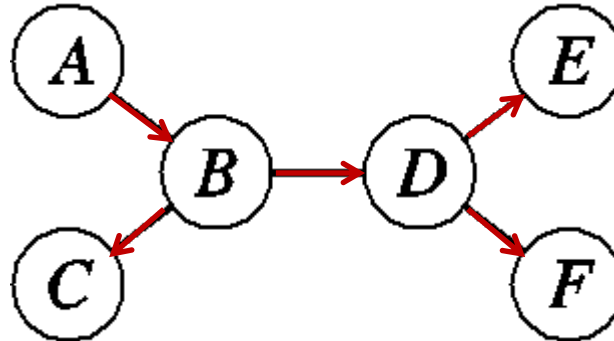
- ▶ Any two variables are connected by only one path
- ▶ Any tree-structured CSP can be solved in time linear in  $n$



# Tree structured CSPs: topological ordering

---

- ▶ Construct a rooted tree (picking any variable to be root, ...)



- ▶ Order variables from root to leaves such that every node's parent precedes it in the ordering (topological ordering)



**function** *TREE\_CSP\_SOLVER*(*csp*) **returns** a solution or failure

**input:** *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  $X$

$X \leftarrow \text{TOPOLOGICAL}(X, \text{root})$

**for**  $j = n$  **down to** 2 **do**

$\text{MAKE\_ARC\_CONSISTENT}(\text{PARENT}(X_j), X_j)$

**if** it cannot be made consistent **then return** *failure*

**for**  $i = 1$  **to**  $n$  **do**

$\text{assignment}[X_i] \leftarrow$  anyconsistent value from  $D_i$

**if** there is no consistent value **then return** *failure*

**return** *assignment*

Why doesn't this algorithm work with loops?

# Tree structured CSP Solver

$X \leftarrow$  Topological Sort

for  $i = n$  downto 2 do

    Make-Arc-Consistent(Parent( $X_i$ ),  $X_i$ )

for  $i = 1$  to  $n$  do

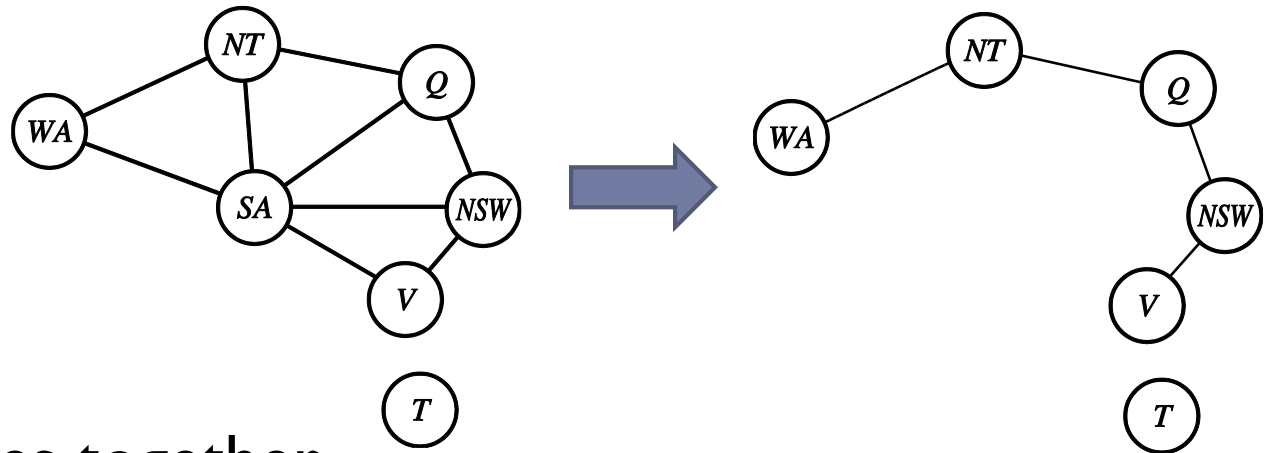
$X_i \leftarrow$  any remaining value in  $D_i$

remove all values from domain of Parent( $X_i$ ) which may violate arc-consistency.

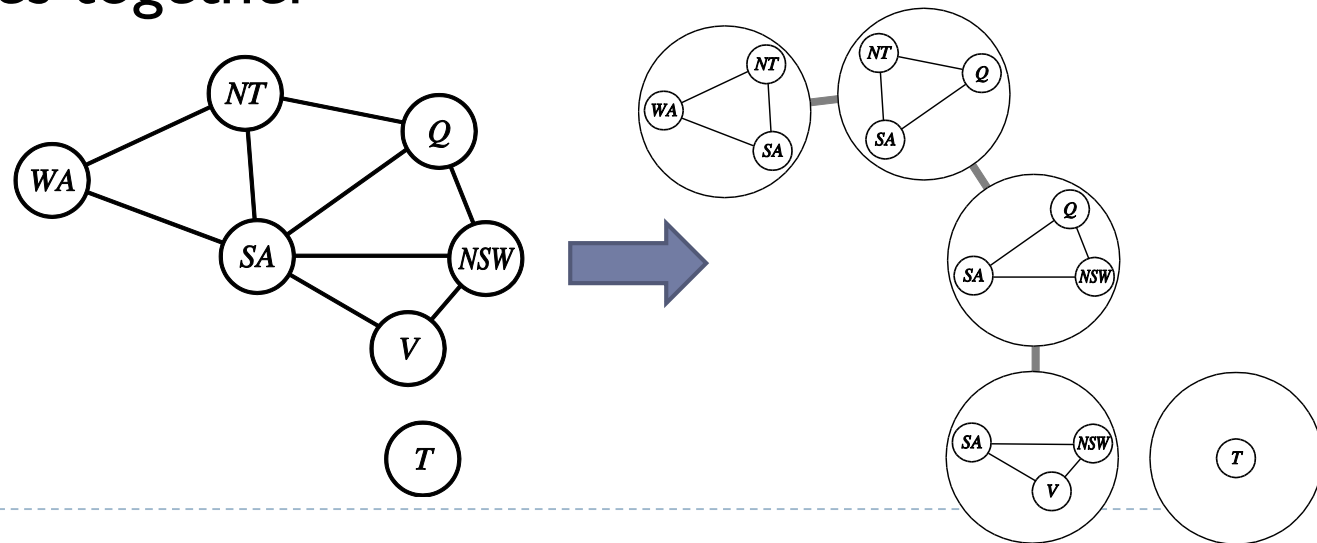
- ▶ After running loop I, any arc from a parent to its child is arc-consistent.
- ▶  $\Rightarrow$  if the constraint graph has no loops, the CSP can be solved in  $O(nd^2)$  time.

# Reduction of general graphs into trees

## ► Removing nodes



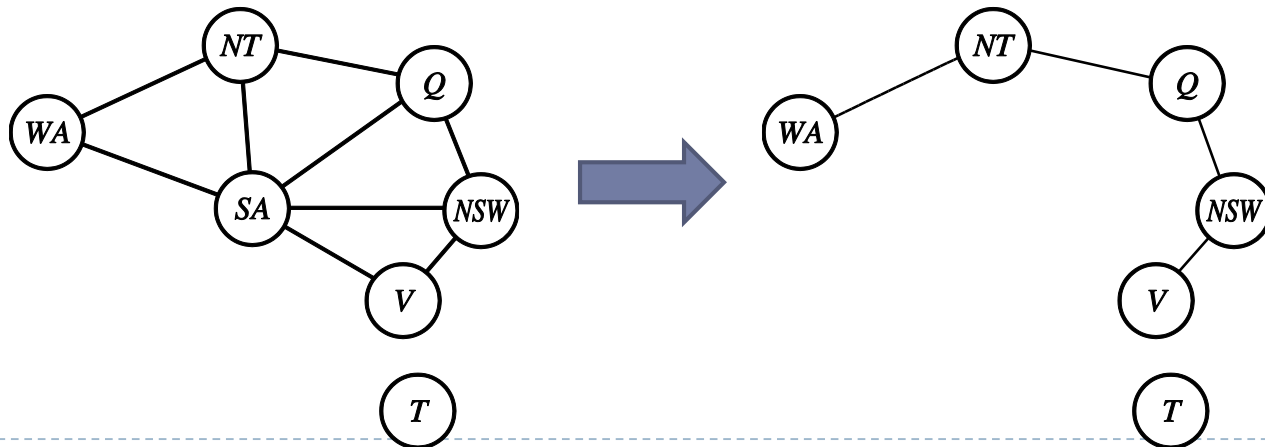
## ► Collapsing nodes together



# Cut-set conditioning

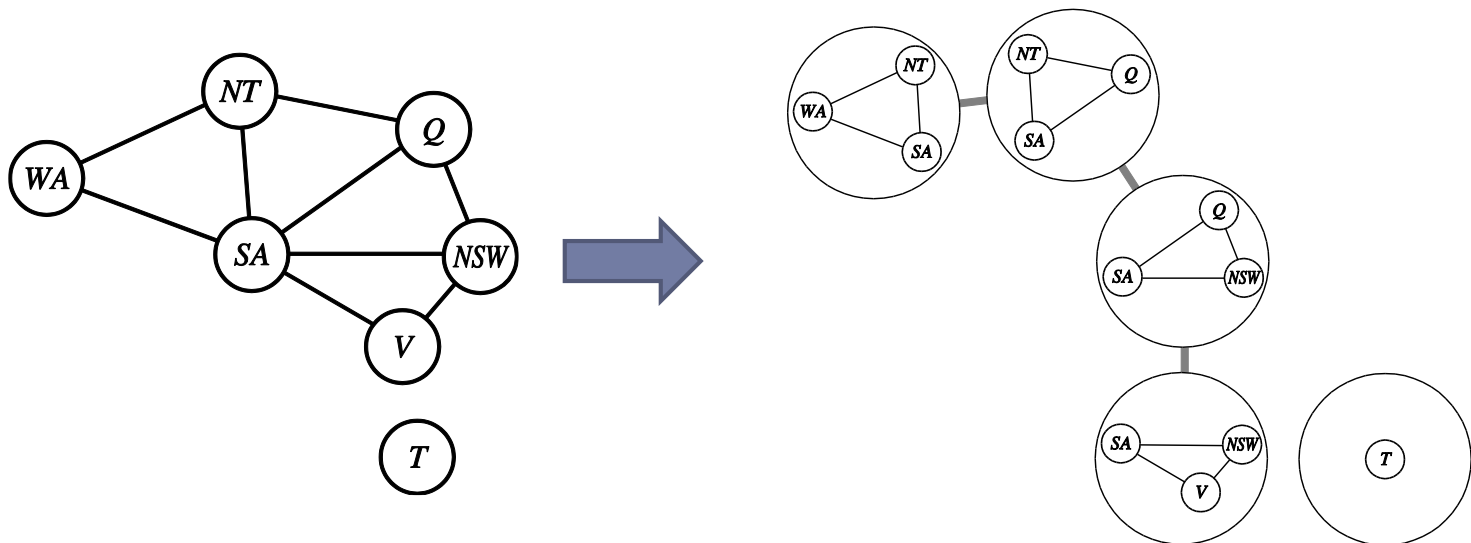
- 1) Find a subset S such that the remaining graph becomes a tree
- 2) For each possible consistent assignment to S
  - a) remove inconsistent values from domains of remaining variables
  - b) solve the remaining CSP which has a tree structure

- ▶ Cutset size  $c$  gives runtime  $O((d^c) (n - c) d^2)$ 
  - ▶ very fast for small  $c$
  - ▶  $c$  can be as large as  $n - 2$



# Tree Decomposition

- ▶ Create a tree-structured graph of overlapping sub-problems (each sub-problem as a mega-variable)
- ▶ Solve each sub-problem (enforcing local constraints)
- ▶ Solve the tree-structured CSP over mega-variables



# Solving CSPs by local search algorithms

---

- ▶ In the CSP formulation as a search problem, path is irrelevant, so we can use complete-state formulation
- ▶ **State**: an assignment of values to variables
- ▶ **Successors( $s$ )**: all states resulted from  $s$  by choosing a new value for a variable
- ▶ **Cost function  $h(s)$** : Number of violated constraints
- ▶ **Global minimum**:  $h(s) = 0$



**function** *MIN\_CONFLICTS*(*csp*, *max\_steps*) **returns** a solution or failure

**inputs:** *csp*, a constraint satisfaction problem

*max\_steps*, the number of steps allowed before giving up

*current*  $\leftarrow$  an initial complete assignment for *csp*

**for** *i* = 1 to *max\_steps* **do**

**if** *current* is a solution for *csp* **then return** *current*

*var*  $\leftarrow$  a randomly chosen conflicted variable from *csp.VARIABLES*

*value*  $\leftarrow$  the value *v* for *var* that minimizes *CONFLICTS*(*var*, *v*, *current*, *csp*)

    set *var* = *value* in *current*

**return** *failure*

if current state is consistent then

    return it

else

    choose a random variable *v*, and change assignment of *v*  
    to a value that causes minimum conflict.

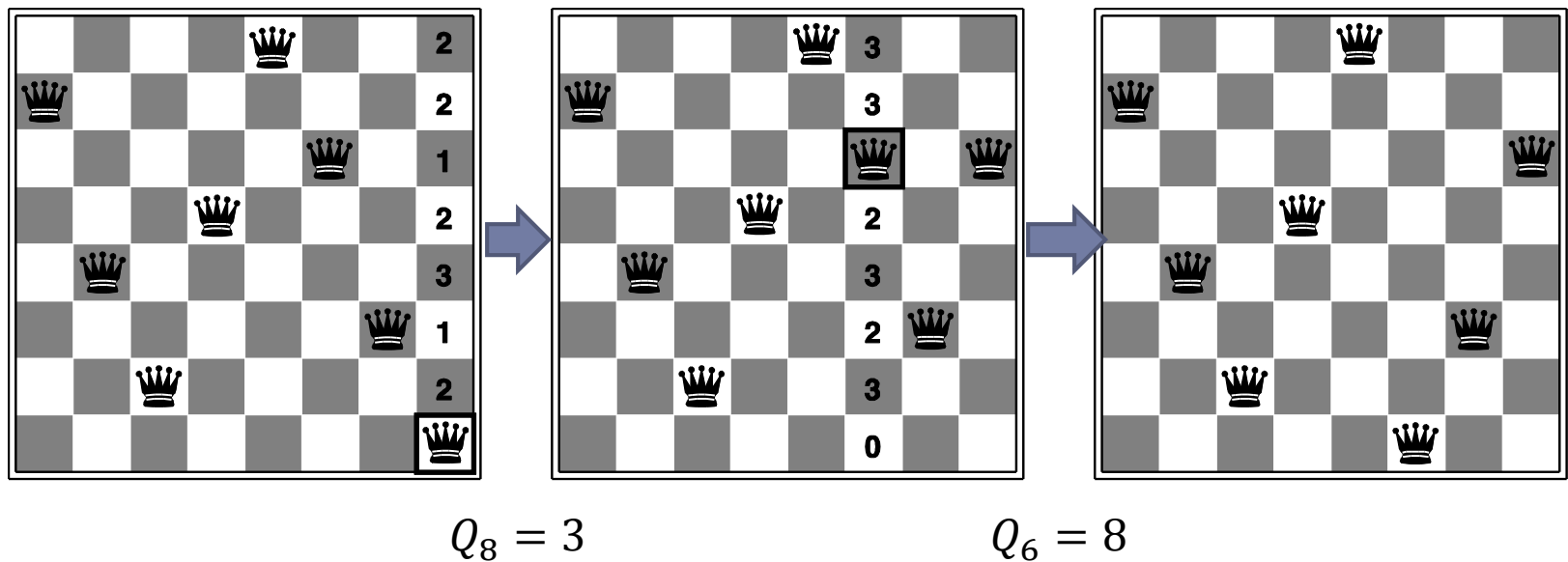
# Local search for CSPs

---

- ▶ Variable selection: **randomly** select any conflicted variable
- ▶ Value selection by **min-conflicts** heuristic
  - ▶ choose value that violates the fewest constraints
    - ▶ i.e., hill-climbing
- ▶ Given random initial state, it can solve  $n$ -queens in almost constant time for arbitrary  $n$  with high probability
  - ▶  $n = 1000000$  in an average of 50 steps
- ▶ N-queens is easy for local search methods (while quite tricky for backtracking)
  - ▶ Solutions are very densely distributed in the space and any initial assignment is guaranteed to have a solution nearby.

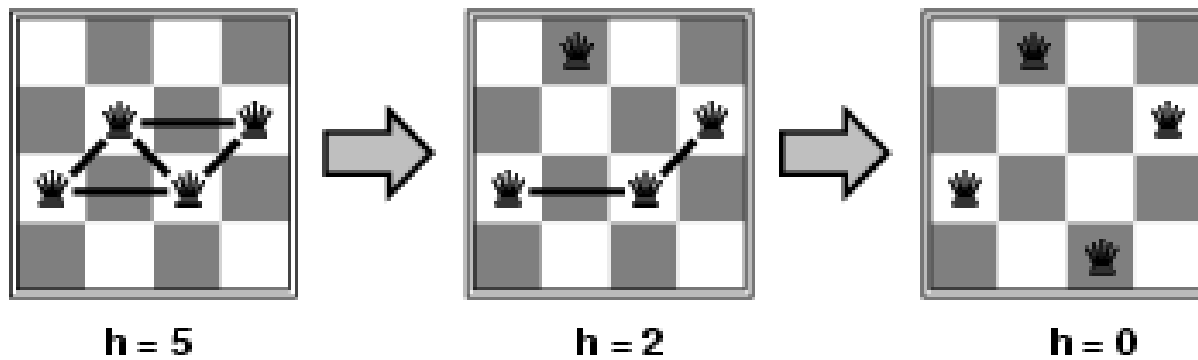
# 8-Queens example

---



# 4-Queens example

---



# Summary

---

- ▶ CSP benefits
  - ▶ Standard representation of many problems
  - ▶ Generic heuristics (no domain specific expertise)
- ▶ CSPs solvers (based on systematic search)
  - ▶ Backtracking
  - ▶ Variable ordering and value selection heuristics
  - ▶ Forward checking prevents one-step more assignments that guarantee later failure.
  - ▶ Constraint propagation: in addition to forward checking propagates constraints (to detect some inconsistencies earlier)
- ▶ Graph structure may be useful in solving CSPs efficiently.
- ▶ Iterative min-conflicts (based on local search) is usually effective in solving CSPs.