

Constraint Satisfaction Problems

Michail G. Lagoudakis

Intelligent Systems Laboratory

School of Electrical and Computer Engineering

Technical University of Crete

Chania, Crete, Greece



Constraint Satisfaction Problem (CSP)

■ CSP Definition

- variables: $X_1, X_2, \dots, X_n$
- (non-empty) domains of possible values: $D_1, D_2, \dots, D_n$
- constraints: $C_1, C_2, \dots, C_m$
 - allowable combinations of values to subsets of variables
- value assignment: $\{X_i = v_i, X_j = v_j, \dots\}$
 - can be partial or complete
- consistent (or legal) assignment: no constraint is violated
- solution: complete and consistent assignment

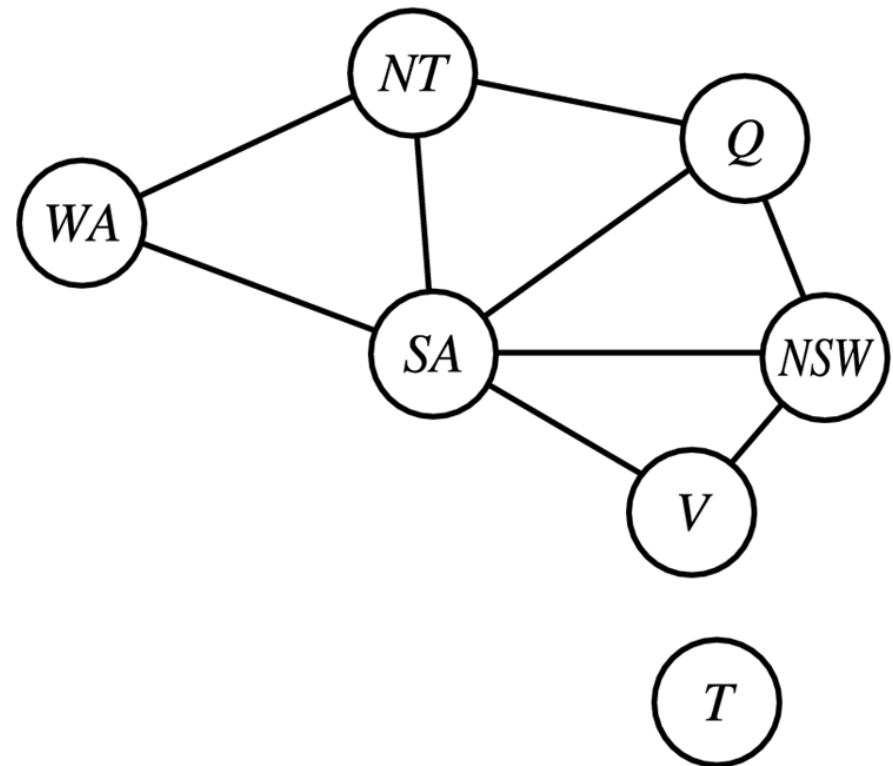
■ Extension

- in addition, optimization of an objective function

Example: Map Coloring



Map of Australia



Constraint Graph

goal: assign colors so that no neighboring regions have the same color

Example: Map Coloring

■ Variables

- *WA, NT, Q, NSW, V, SA, T*

■ Domains

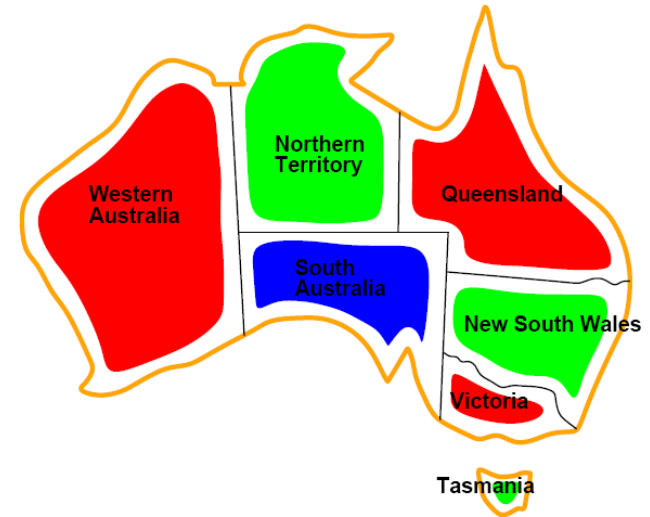
- {*red, green, blue*} for all variables

■ Constraints

- neighboring regions cannot have the same color
- representation by enumeration of allowable combinations:
 $\{ (red, green), (red, blue), (green, red), (green, blue), (blue, red), (blue, green) \}$
- representation by using specialized notation: $WA \neq NT, \dots$

■ Solution (one of many)

- $WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green$



Benefits

■ Advantages

- generic, formal language of representation
- general-purpose algorithms
- utilization of the structure of problems

■ Applications

- task scheduling
- curriculum development and course assignment
- semester timetable for a university
- coloring graph
- hardware configuration

Constraint Satisfaction as Search I

■ States

- partial or complete (consistent) value assignments to variables
- initial state: empty value assignment

■ Successor function

- value assignment that does not cause any conflict

■ Goal state

- complete (consistent) value assignment to variables

■ Path cost

- irrelevant (solution is independent of the search path)

■ Algorithm

- depth-first search (maximum depth = number of variables)

Constraint Satisfaction as Search II

■ States

- complete value assignments to variables
- initial state: random value assignment (possible conflicts)

■ Local Moves

- new value assignment to some variable to reduce conflicts

■ Goal state

- (complete) consistent value assignment to variables

■ Path cost

- irrelevant (solution is independent of the search path)

■ Algorithm

- local search

Problem Characteristics

■ Formulation

- incremental (depth-first search)
- complete states (local search)

■ Variables

- discrete / continuous variables
- finite / infinite domains of values

■ Constraints

- unary / binary / ternary / n -ary (higher-order)
- soft constraints
 - can be violated, but prefer to satisfy
 - reduction to optimization with weights

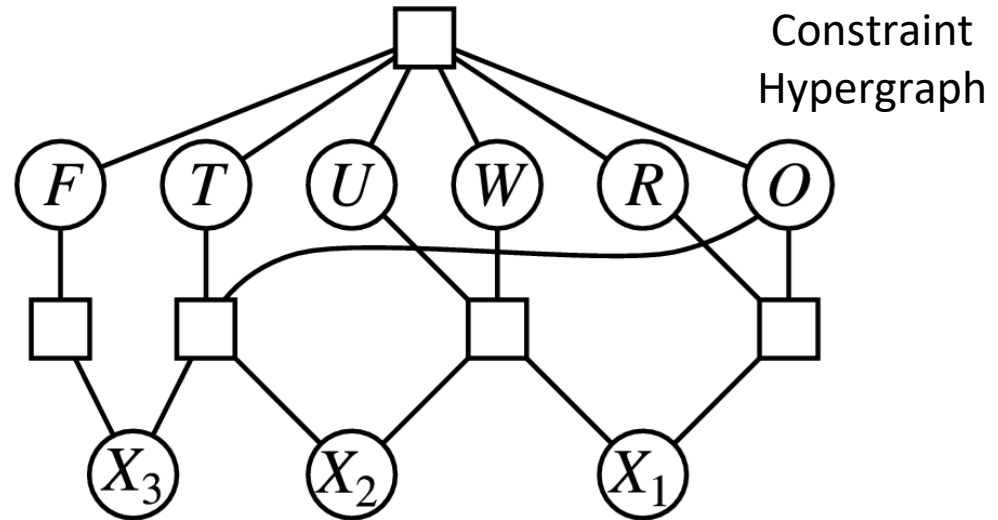
Higher-Order Constraints

Cryptarithmic Puzzle

$$\begin{array}{r}
 765 \\
 + 765 \\
 \hline
 1530
 \end{array}
 \quad
 \begin{array}{r}
 TWR \\
 + TWR \\
 \hline
 FOUR
 \end{array}$$

$$\begin{array}{r}
 132 \\
 + 132 \\
 \hline
 0264
 \end{array}$$

$$\begin{array}{r}
 734 \\
 + 734 \\
 \hline
 1468
 \end{array}$$

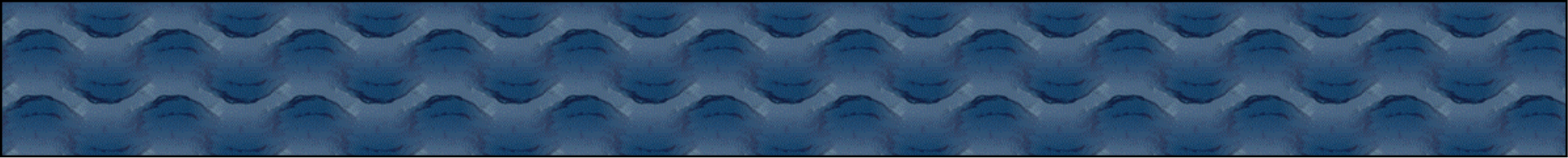


Variables

- F, T, U, W, R, O
 - $D = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- X_1, X_2, X_3
 - $D = \{0, 1\}$

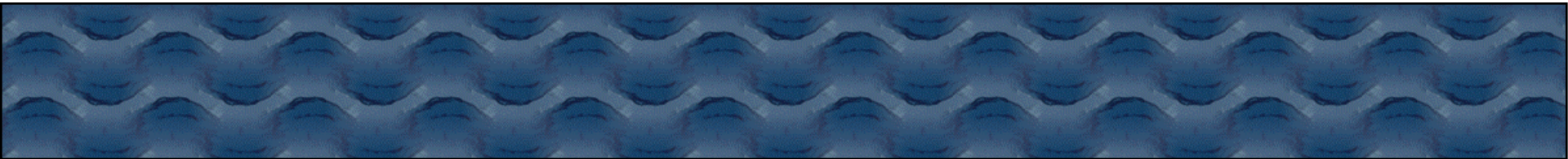
Constraints

- $Alldiff(F, T, U, W, R, O)$
- $O + O = R + 10 \times X_1$
- $X_1 + W + W = U + 10 \times X_2$
- $X_2 + T + T = O + 10 \times X_3$
- $X_3 = F$



Backtracking Search

Backtracking at Dead-Ends



Commutativity and Backtracking

■ Naive search

- depth-first search
- $b = nd, (n-1)d, (n-2)d, \dots \Rightarrow n!d^n$ leaves, but only d^n assignments!

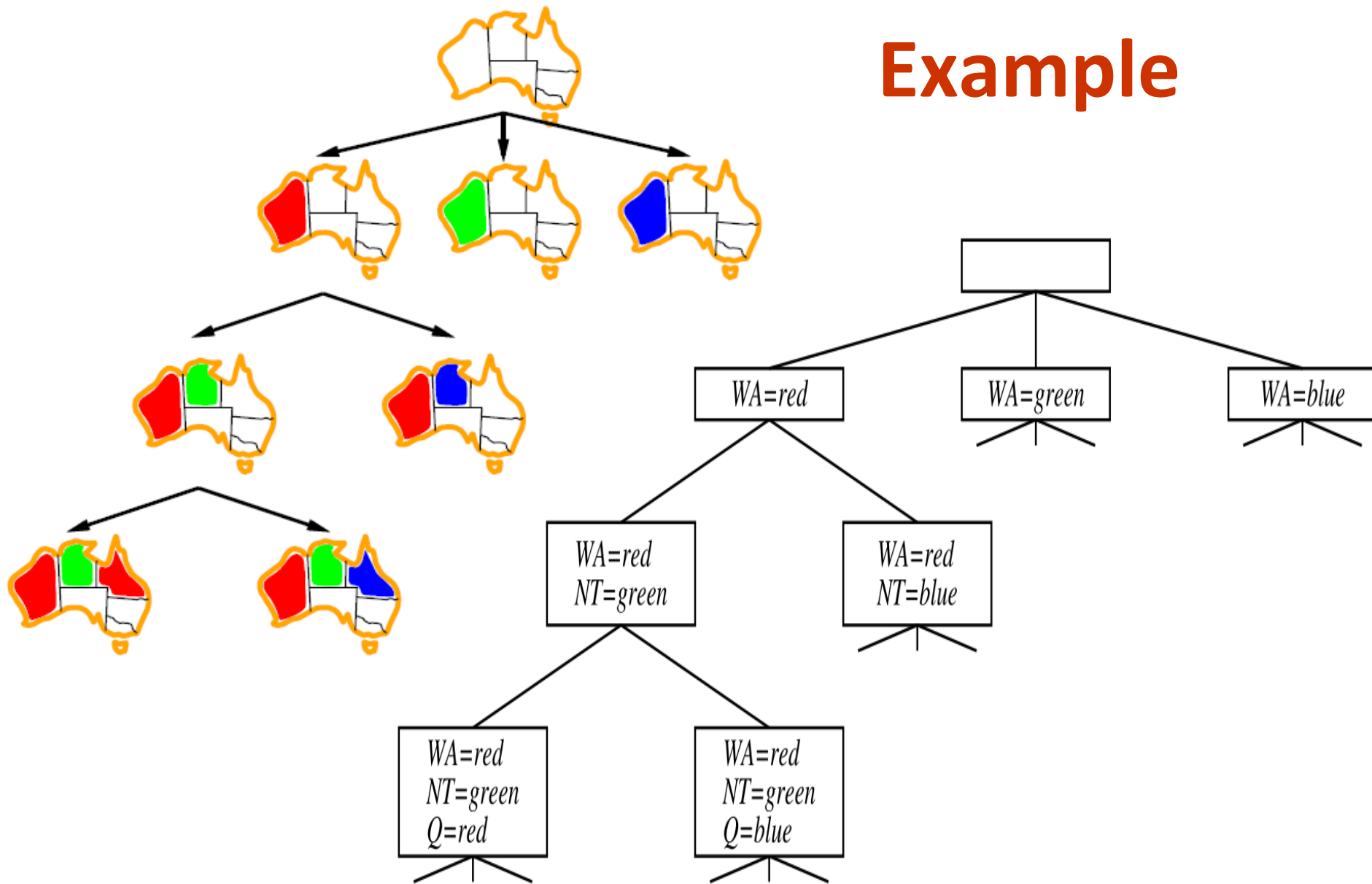
■ Commutativity

- the ordering of actions (assignments) does not affect the result
- assignment to a *single* variable per level $\Rightarrow b=d \Rightarrow d^n$ leaves

■ Backtracking search

- depth-first search
- assignment to a *single* variable per level
- variables can be ordered arbitrarily or heuristically
- backtracking when some variable has no legal values left

Example



Backtracking Search Algorithm

```
function BACKTRACKING-SEARCH(csp) returns solution/failure
  return RECURSIVE-BACKTRACKING({ }, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
    if value is consistent with assignment given CONSTRAINTS[csp] then
      add {var = value} to assignment
      result ← RECURSIVE-BACKTRACKING(assignment, csp)
      if result ≠ failure then return result
      remove {var = value} from assignment
  return failure
```

Improvements

■ Conventional search

- use of heuristic functions
- exploitation of domain knowledge

■ Constraint Satisfaction Problems

- use of generic techniques
 - in which variable will the next assignment be made?
 - in what order will the legal values be assigned?
 - what are the effects of the current assignment to the other variables?
 - how can a definite failure be detected early?
 - how can we avoid repeated failures?
 - how can we exploit the structure of the problem?
 - ...

Variable and Value Ordering

var $\leftarrow$ SELECT-UNASSIGNED-VARIABLE(VARIABLES[*csp*], *assignment*, *csp*)

- **Variable selection: the most-constrained**

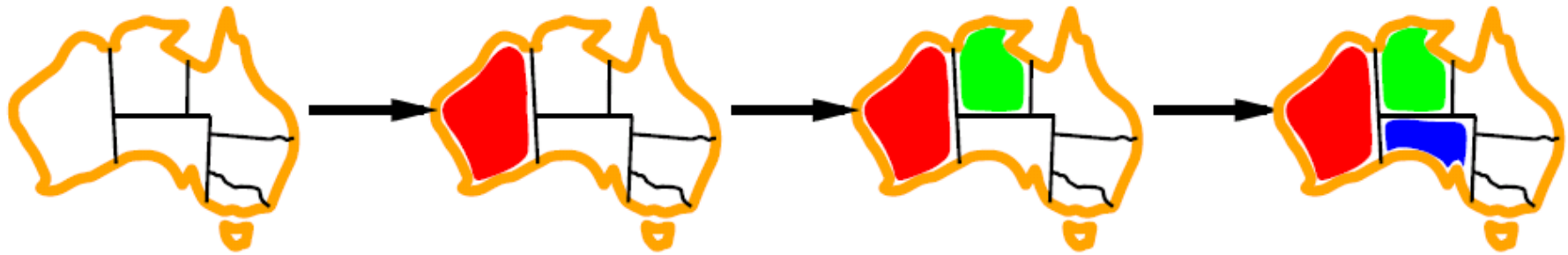
- minimum remaining values (MRV) heuristic
- degree heuristic

for each *value* **in** ORDER-DOMAIN-VALUES(*var*, *assignment*, *csp*) **do**

- **Value selection: the least-constraining**

- least-constraining value (LCV) heuristic

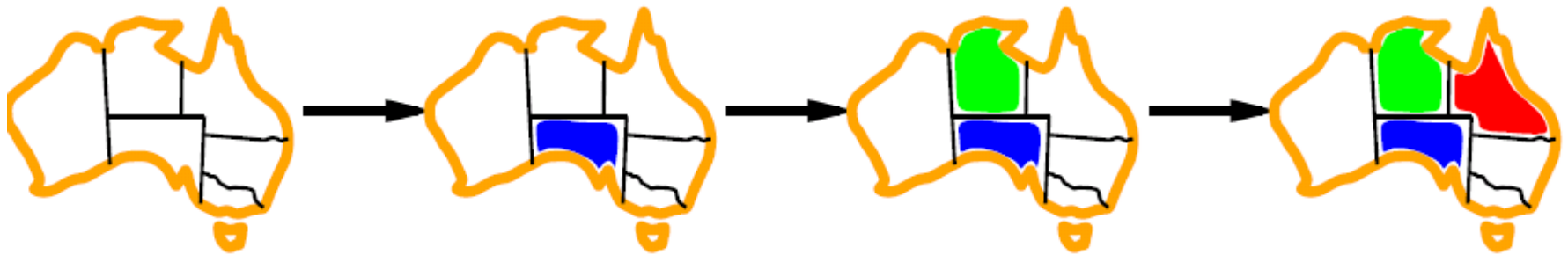
Minimum Remaining Values Heuristic



■ Heuristic mechanism

- rule: *choose the variable with the fewest legal values*
- known also as the “most-constrained variable” heuristic
- known also as the “fail-first” heuristic
- attempts to detect a possible failure early

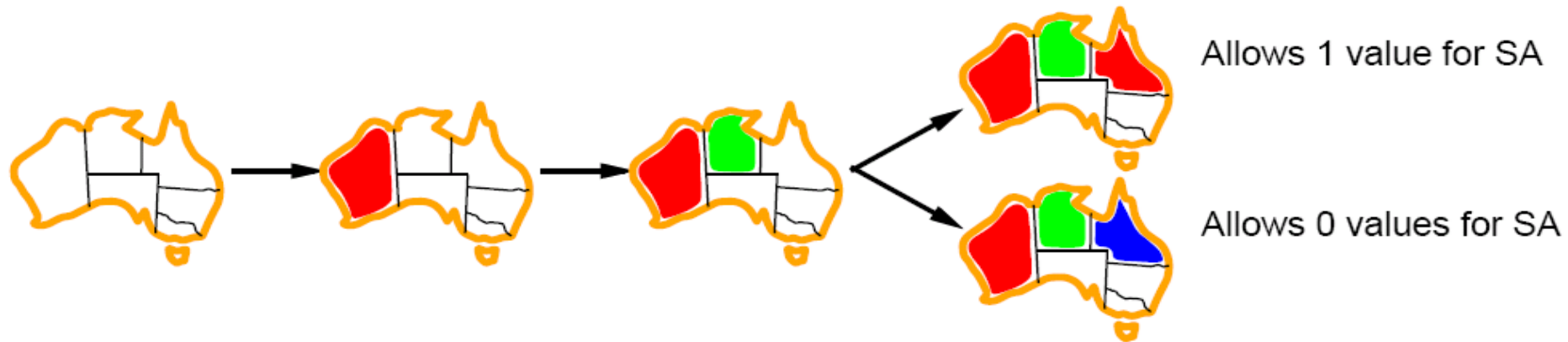
Degree Heuristic



■ Heuristic mechanism

- rule: *choose the variable involved in most constraints compared to the other unassigned variables*
- it relates to node degree in the constraint graph
- attempts to reduce the branching factor in future choices
- acts as tie-breaker in case of an equal number of legal values

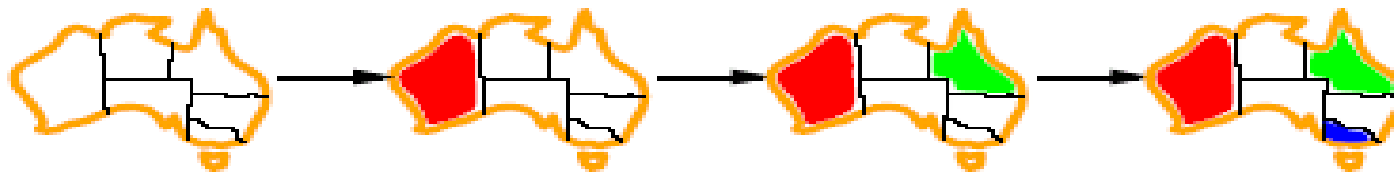
Least-Constraining Value Heuristic



■ Heuristic mechanism

- rule: *choose a value for the variable that excludes the fewest options for neighboring free variables*
- attempts to allow maximum flexibility in the next steps
- useful when we ask for only *one* solution, not all solutions

Forward Checking

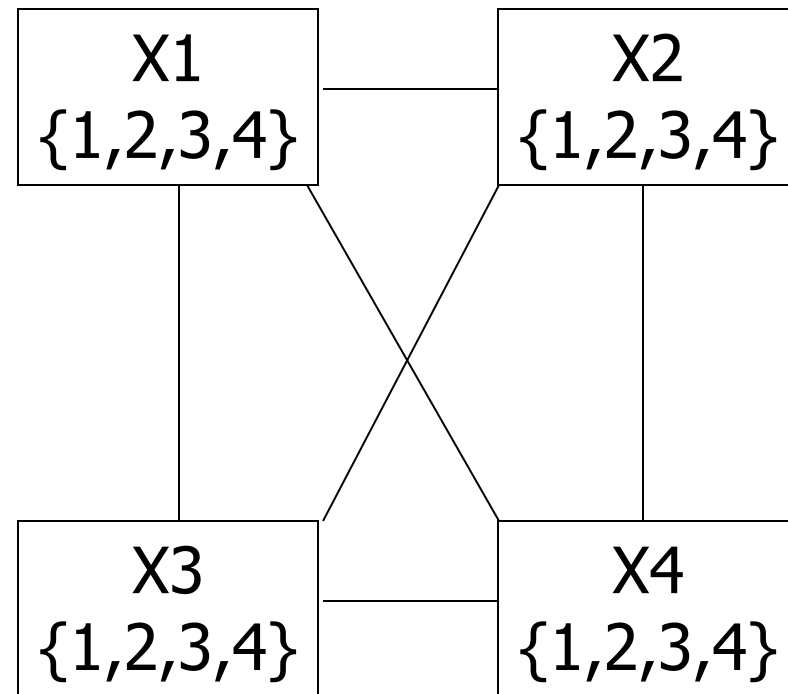


WA	NT	Q	NSW	V	SA	T

- rule: *when assigning a value to a variable X , we check the variables associated with X and remove values from their domains*
- propagation of information from bound to free variables
- makes better use of constraints

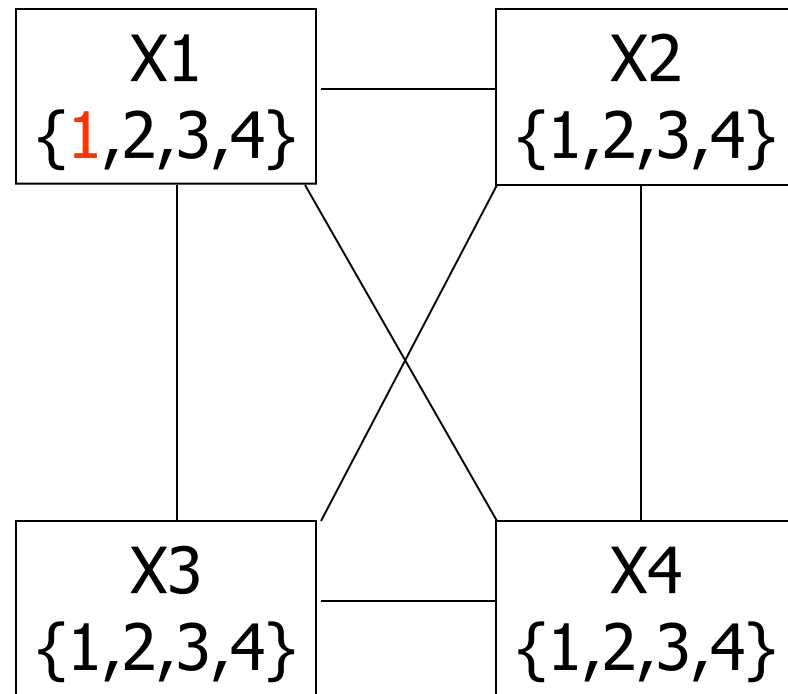
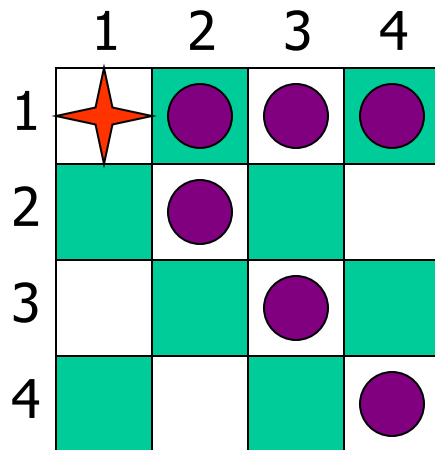
Example: 4-Queens Problem

	1	2	3	4
1				
2				
3				
4				

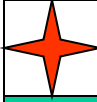
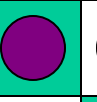
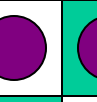
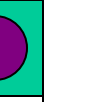
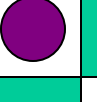


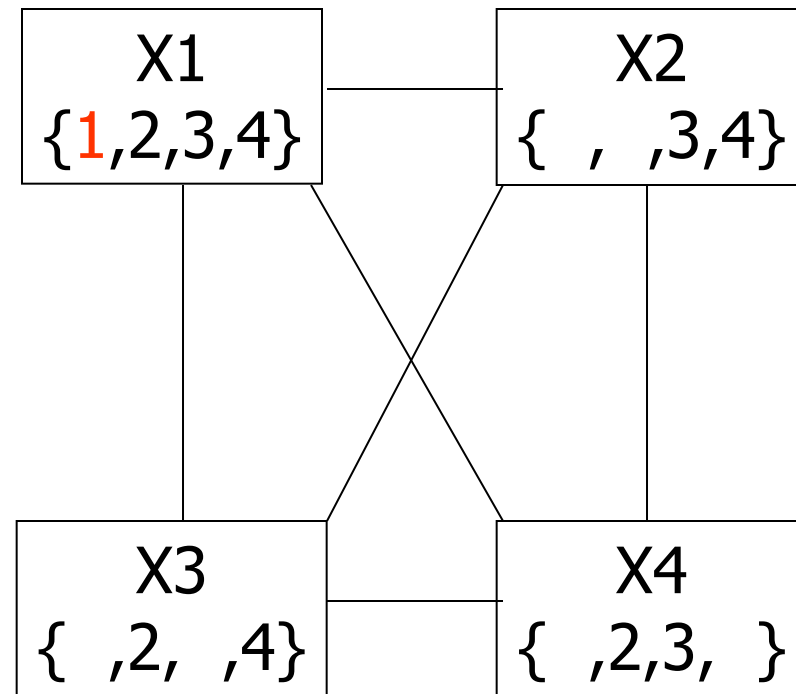
[these slides are taken from the CMSC 421 course of B. J. Dorr]

Example: 4-Queens Problem



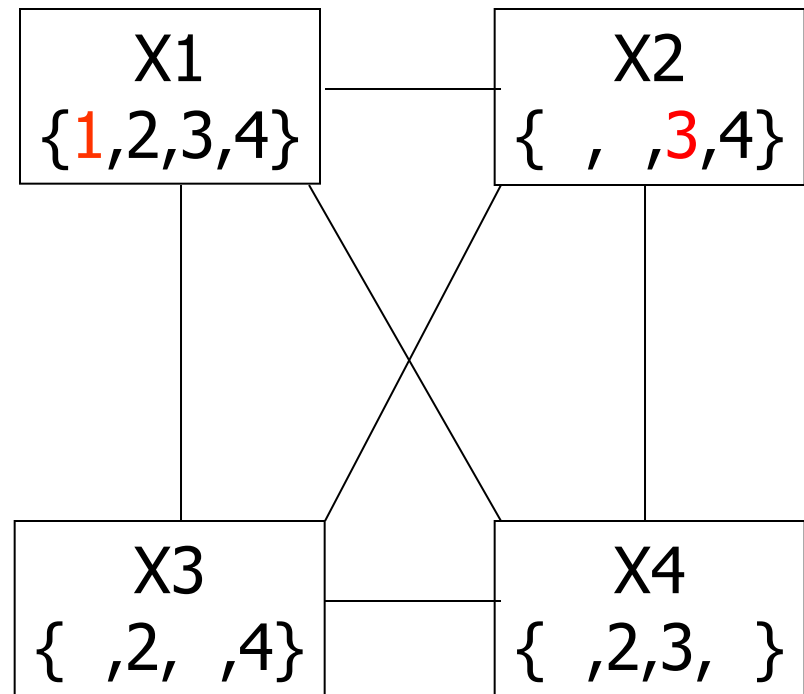
Example: 4-Queens Problem

	1	2	3	4
1				
2				
3				
4				

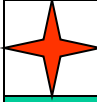
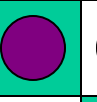
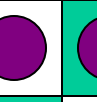
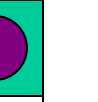
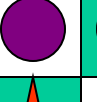
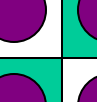


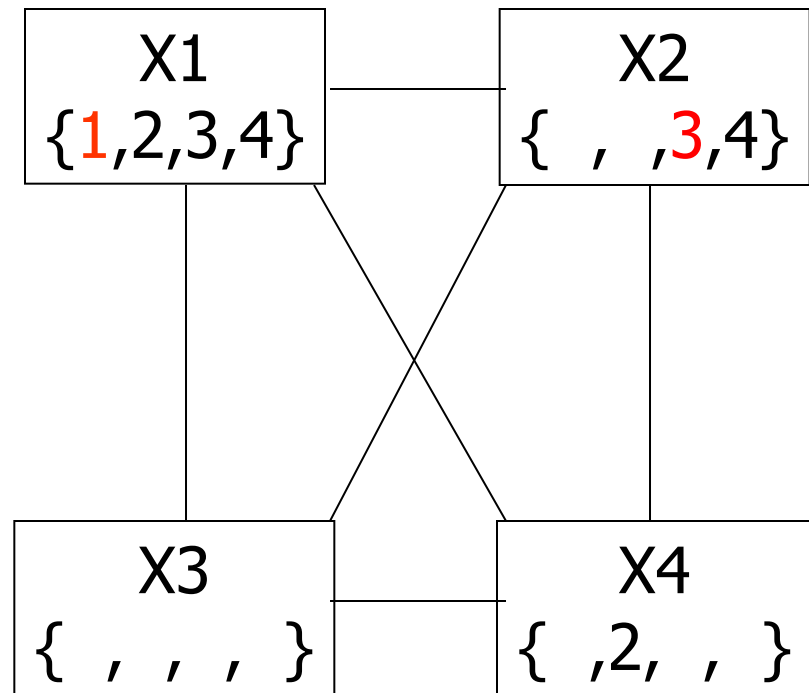
Example: 4-Queens Problem

	1	2	3	4
1	★	●	●	●
2		●	●	
3		★	●	●
4			●	●

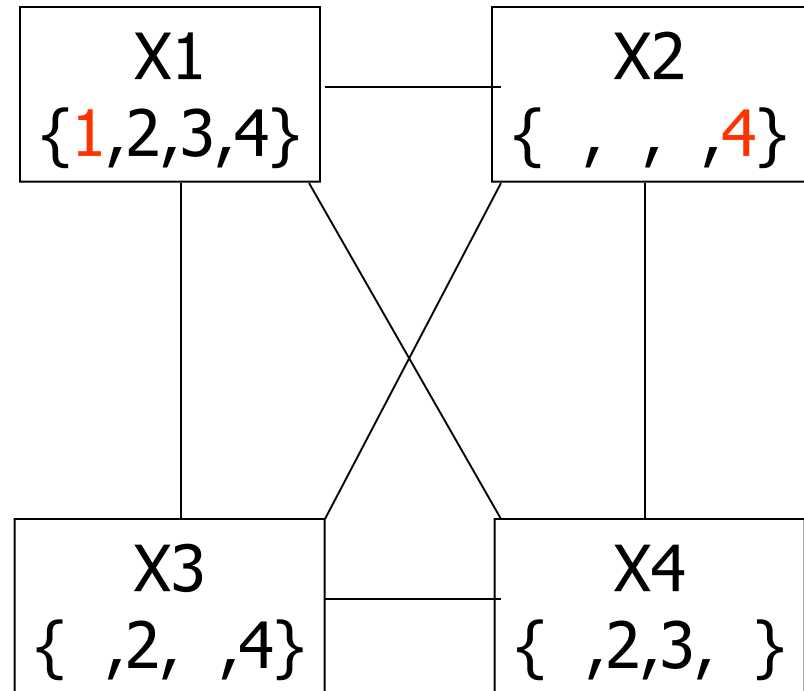
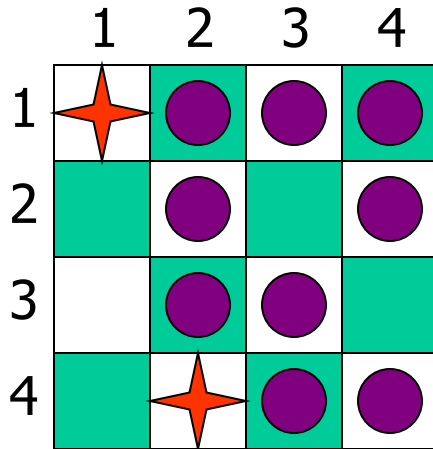


Example: 4-Queens Problem

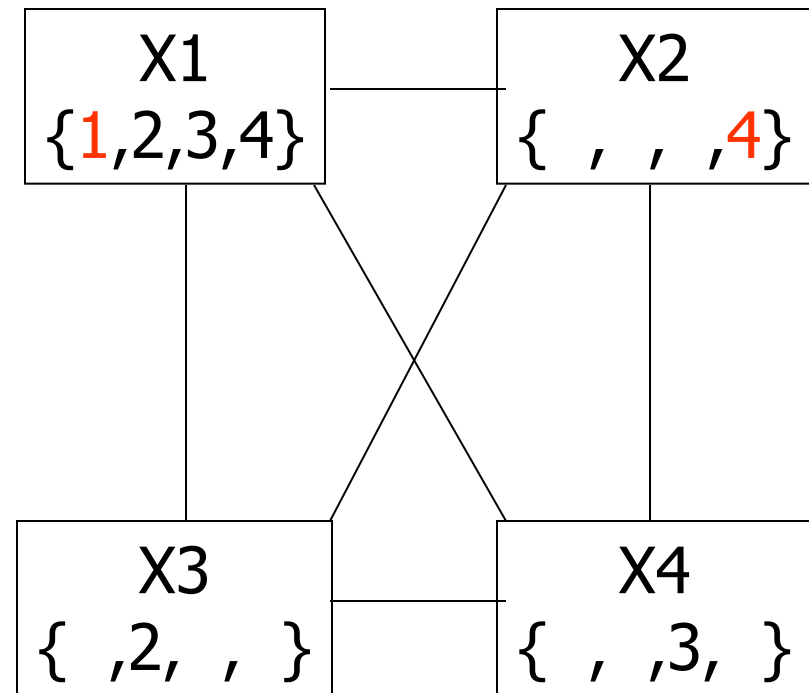
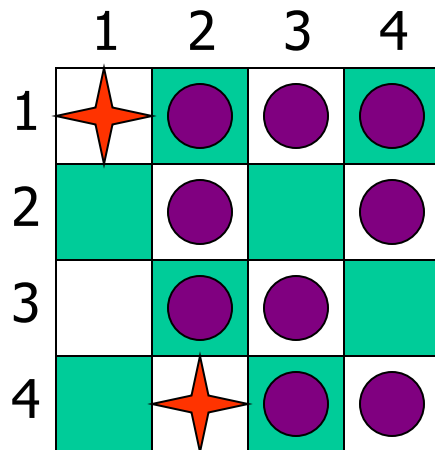
	1	2	3	4
1				
2				
3				
4				



Example: 4-Queens Problem

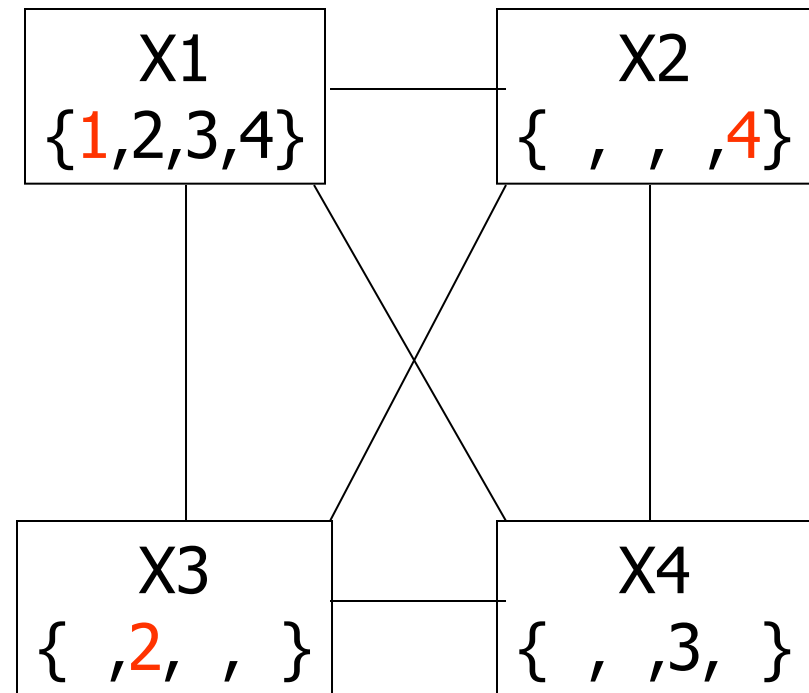


Example: 4-Queens Problem

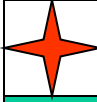
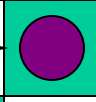
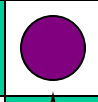
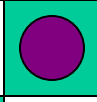
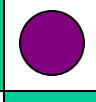
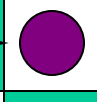
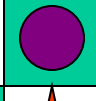
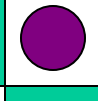
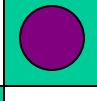
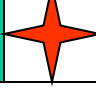
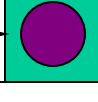
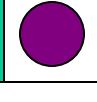


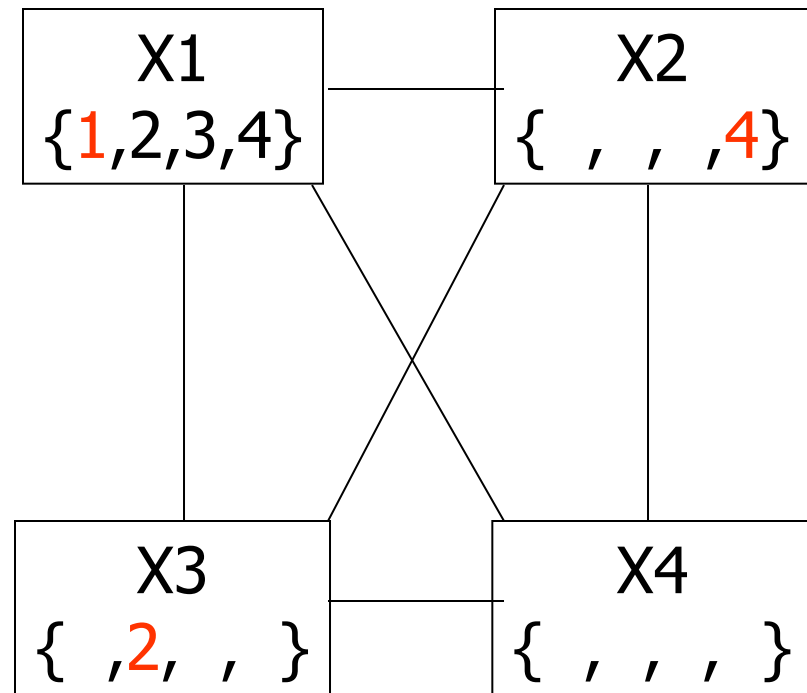
Example: 4-Queens Problem

	1	2	3	4
1	★	●	●	●
2		●	★	●
3		●	●	●
4		★	●	●



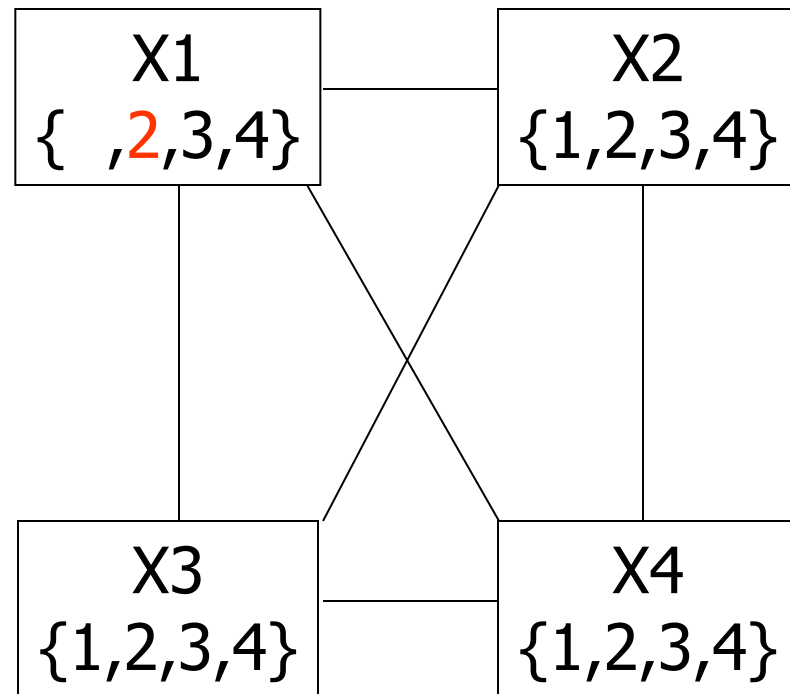
Example: 4-Queens Problem

	1	2	3	4
1				
2				
3				
4				



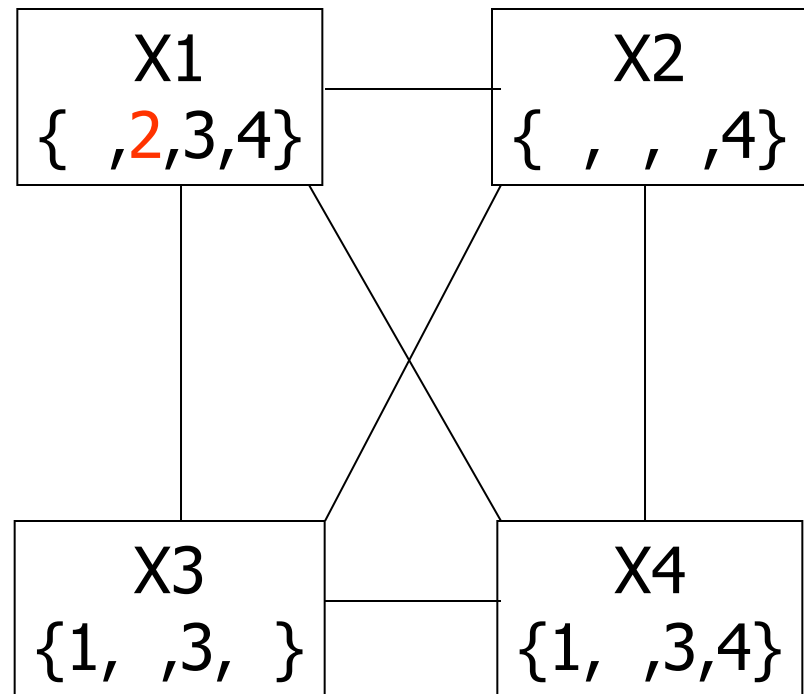
Example: 4-Queens Problem

	1	2	3	4
1	●	●		
2	★	●	●	●
3		●		
4			●	



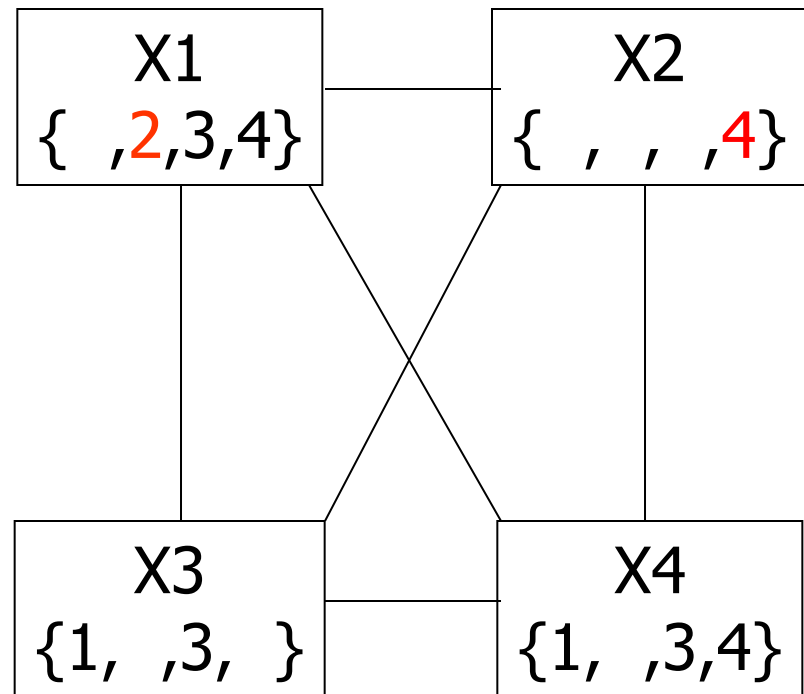
Example: 4-Queens Problem

	1	2	3	4
1				
2				
3				
4				



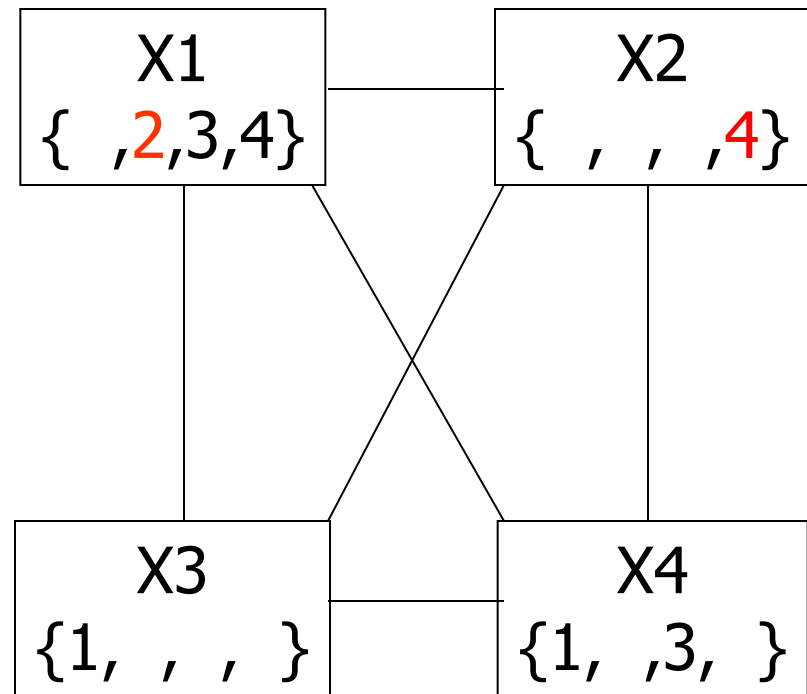
Example: 4-Queens Problem

	1	2	3	4
1				
2				
3				
4				



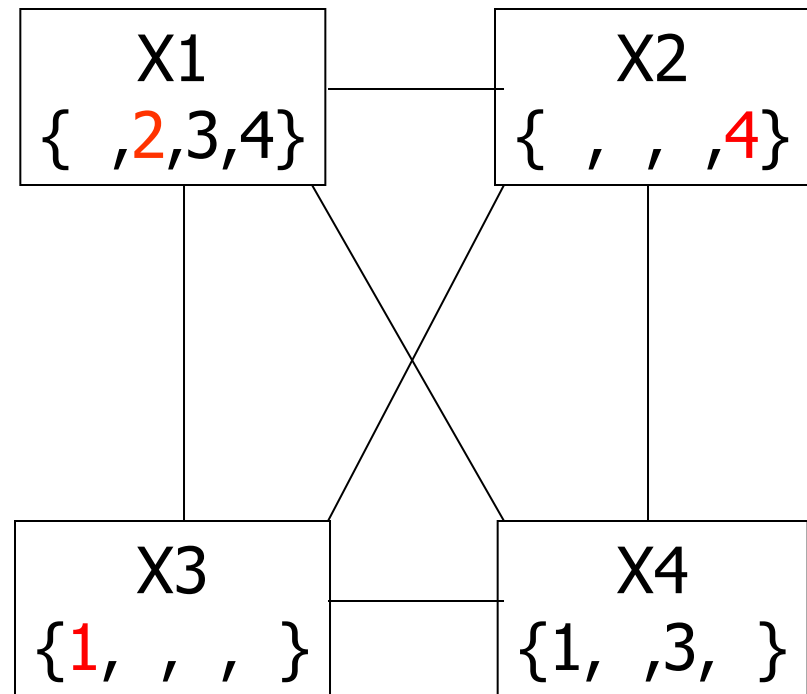
Example: 4-Queens Problem

	1	2	3	4
1				
2				
3				
4				



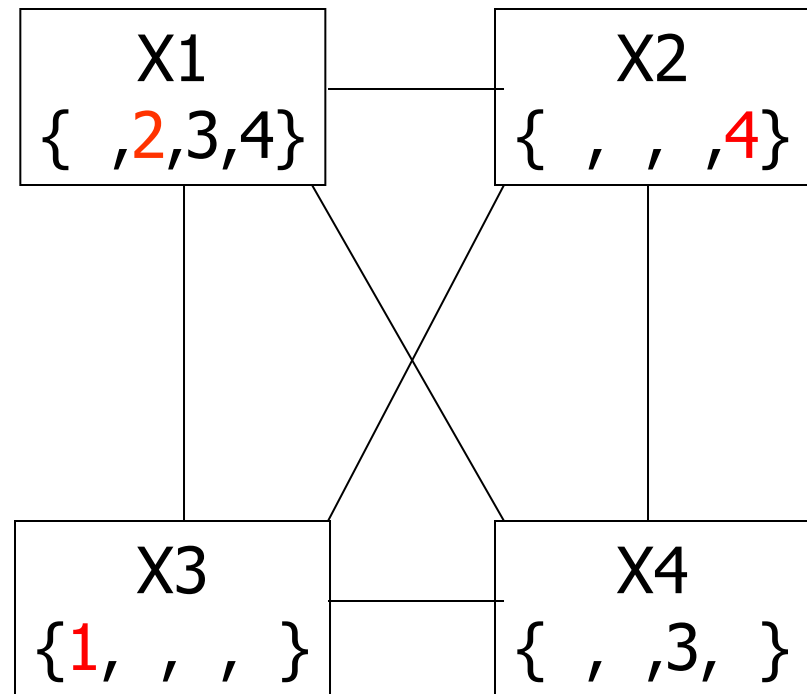
Example: 4-Queens Problem

	1	2	3	4
1				
2				
3				
4				



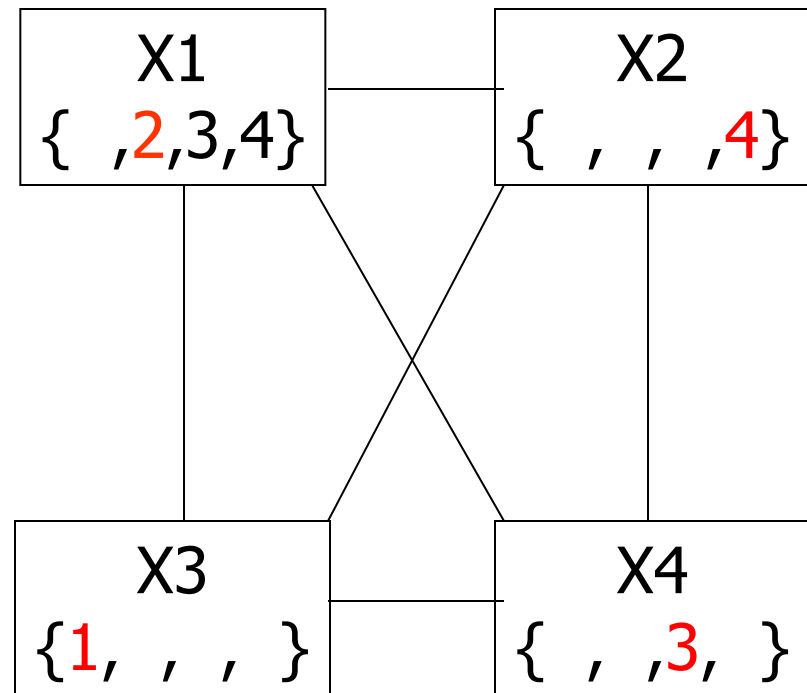
Example: 4-Queens Problem

	1	2	3	4
1				
2				
3				
4				



Example: 4-Queens Problem

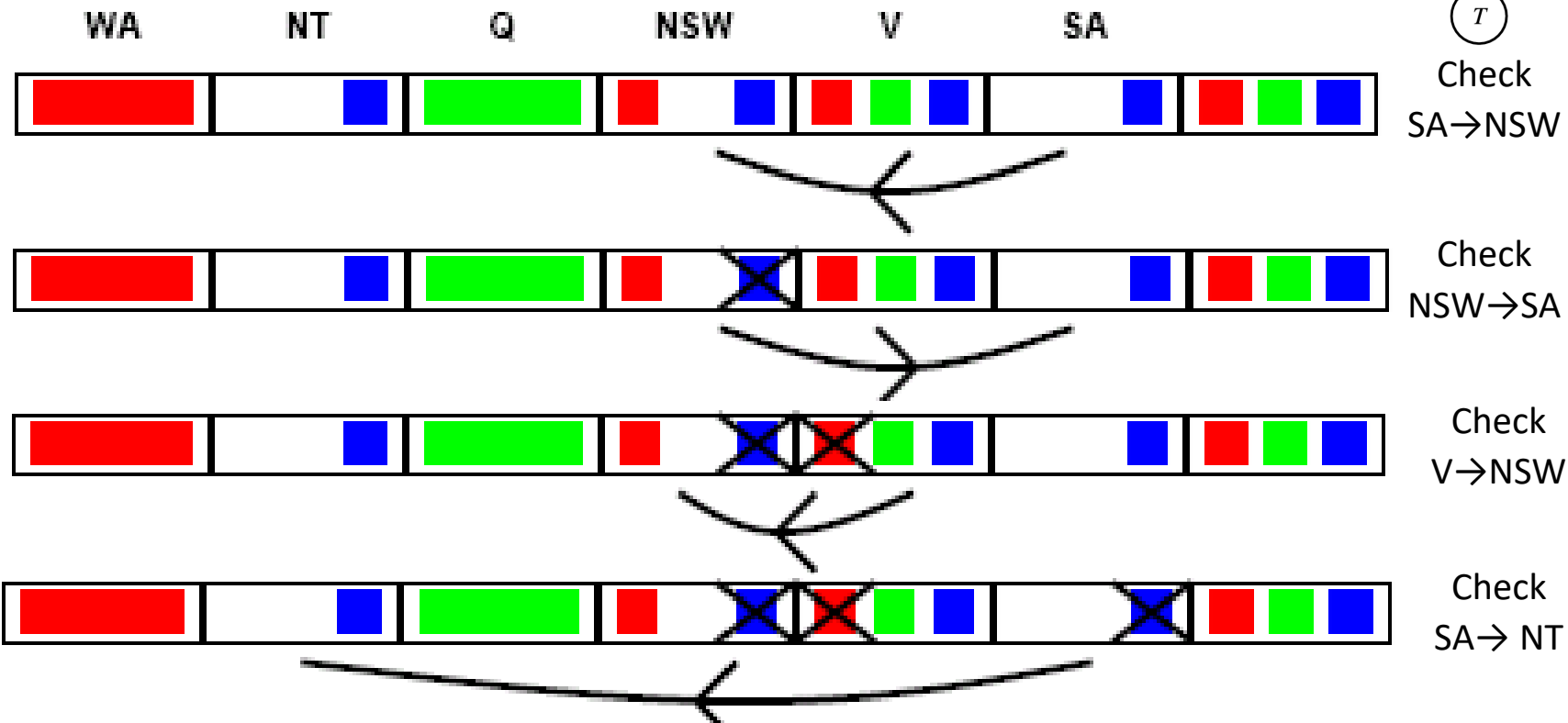
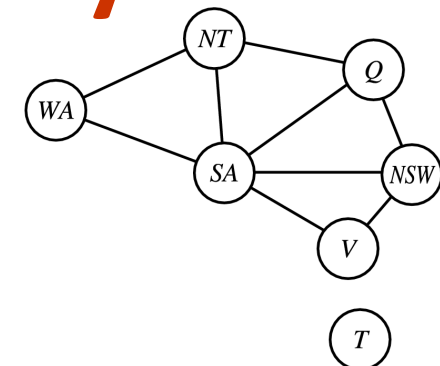
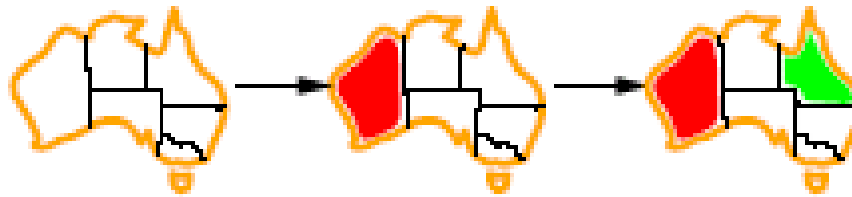
	1	2	3	4
1				
2				
3				
4				



Constraint Propagation

- recursive extension of forward checking to variables for which a value removal occurred at a connected variable
- **arc consistency, 2-consistency**
 - the arc $X \rightarrow Y$ (from the binary constraint $X-Y$) is consistent, if for *each* value of X there exists *some* legal value of Y
 - values of X that lead to 2-inconsistency are removed from X
- **k -consistency**
 - for any value assignment to some set of $k-1$ variables, there exists some legal value for the k -th variable
- **strong k -consistency**
 - k -consistency **and** $(k-1)$ -consistency **and** $(k-2)$ -... **and** 1-consistency

Example: Arc Consistency



Arc Consistency Algorithm

function AC-3(*csp*) **returns** the CSP, possibly with reduced domains

inputs: *csp*, a binary CSP with variables $\{X_1, X_2, \dots, X_n\}$

local variables: *queue*, a queue of arcs, initially all the arcs in *csp*

while *queue* is not empty **do**

$(X_i, X_j) \leftarrow \text{REMOVE-FIRST}(\textit{queue})$

if REMOVE-INCONSISTENT-VALUES(X_i, X_j) **then**

for each X_k **in** NEIGHBORS[X_i] **do**

 add (X_k, X_i) to *queue*

function REMOVE-INCONSISTENT-VALUES(X_i, X_j) **returns** true iff succeeds

removed $\leftarrow$ false

for each x **in** DOMAIN[X_i] **do**

if no value y in DOMAIN[X_j] allows (x, y) to satisfy the constraint $X_i \leftrightarrow X_j$

then delete x from DOMAIN[X_i]; *removed* $\leftarrow$ true

return *removed*

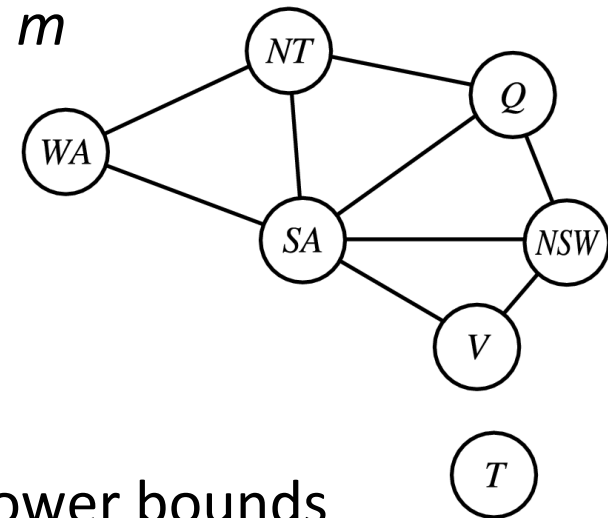
Special Constraints

■ *AllDiff* constraint

- count the number of different values for the variables
- *AllDiff*(NT, SA, Q) inconsistent with {WA=red, NSW=red}

■ Resource constraint: *atmost*

- $atmost(m, X_1, X_2, \dots, X_n) : X_1 + X_2 + \dots + X_n \leq m$
- e.g. *atmost*(10, A, B, C, D):
 - if each variable has domain {3,4,5,6}, the problem is inconsistent
 - if each variable has domain {2,3,4,5,6}, values 5 and 6 can be safely removed



■ Bound propagation

- consistency with respect to upper and lower bounds
 - Flight271 ∈ [0,165], Flight272 ∈ [0,385], Flight271+Flight272 ∈ [420,420]
 - new bounds after propagation: Flight271 ∈ [35,165], Flight272 ∈ [255,385]

Backjumping

■ Conflict set

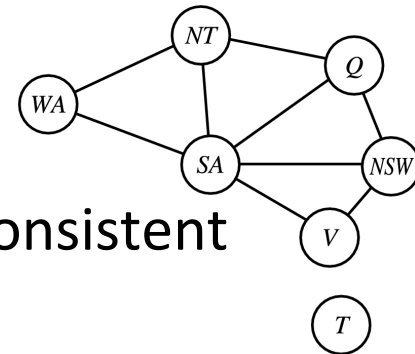
- for each variable Y , it is the set of variables with assigned values, which are connected to Y through constraints (can be built easily during forward checking)

■ Backtracking methods from variable Y

- chronological backtracking, if all assignments were tried and failed
 - back to the most recent variable (the one just before Y)
- backjumping, if all value assignments are inconsistent
 - back to the most recent variable of the conflict set of Y

■ Example

- $\{Q = \text{red}, \text{NSW} = \text{green}, V = \text{blue}, T = \text{red}\}$
- current step: value assignment to SA , all values inconsistent
- conflict set $SA = \{Q, \text{NSW}, V\}$, backjumping to V



Conflict-Directed Backjumping

- backjumping: when values are inconsistent with current assignment
- same thing could have been detected by forward checking
- backjumping is redundant when forward checking is applied

■ Conflict-Directed Backjumping

- assume value inconsistency at Y with conflict set $conflict(Y)$
- backjumping from Y to the most recent variable Z of $conflict(Y)$
- update to Z : $conflict(Z) \leftarrow conflict(Z) \cup conflict(Y) - \{Z\}$
- backjumping even from Z based on $conflict(Z)$ on failure!

■ Characteristics

- never chronological backtracking, even on failure
- $conflict(Y)$: set of variables responsible for inconsistency or failure

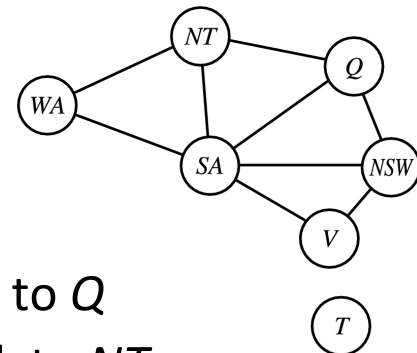
Example: Conflict-Directed Backjumping

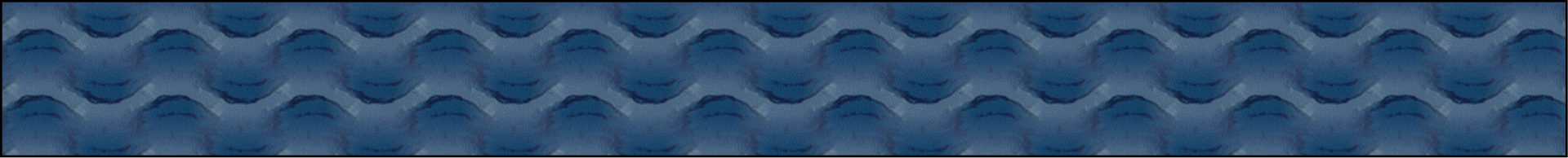
■ Conflict-Directed Backjumping

- assume value inconsistency at Y with conflict set $\text{conflict}(Y)$
- backjumping from Y to the most recent variable Z of $\text{conflict}(Y)$
- update to Z : $\text{conflict}(Z) \leftarrow \text{conflict}(Z) \cup \text{conflict}(Y) - \{Z\}$
- backjumping even from Z based on $\text{conflict}(Z)$ on failure!

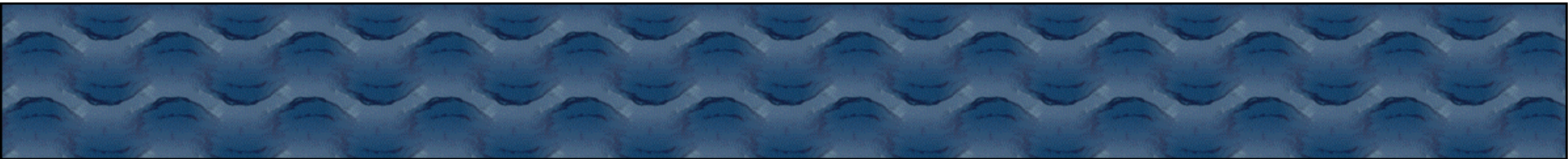
■ Example

- current assignment $\{WA=red, NSW=red, T=red\}$
- variable ordering: WA, NSW, T, NT, Q, SA, V
 - $\text{conflict}(SA)=\{WA, NSW, NT, Q\}$, inconsistency at SA , back to Q
 - $\text{conflict}(Q)=\{NSW, NT\} \Rightarrow \{WA, NSW, NT\}$, failure at Q , back to NT
 - $\text{conflict}(NT)=\{WA\} \Rightarrow \{WA, NSW\}$, failure at NT , back to NSW (!!!)
 - the source of failure was correctly detected from NT afterwards
- simple backjumping: after failure at NT , back to T ...





Local Search for CSPs



Local Search for CSPs

■ States

- complete value assignments to variables
- possible constraint violations

■ Local moves

- assignment of a new value to some variable

■ Choices

- variable: random choice
 - among the variables involved in conflicts
- value: minimum conflict
 - value that leads to minimum number of conflicts

Min-Conflicts Algorithm

function MIN-CONFLICTS(*csp*, *max_steps*) **returns** 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 VARIABLES[*csp*]

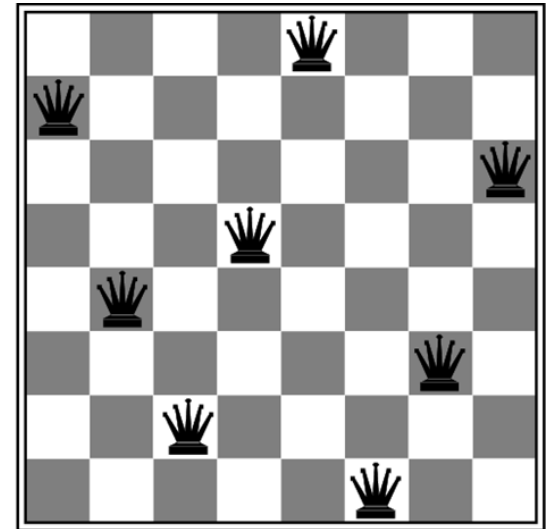
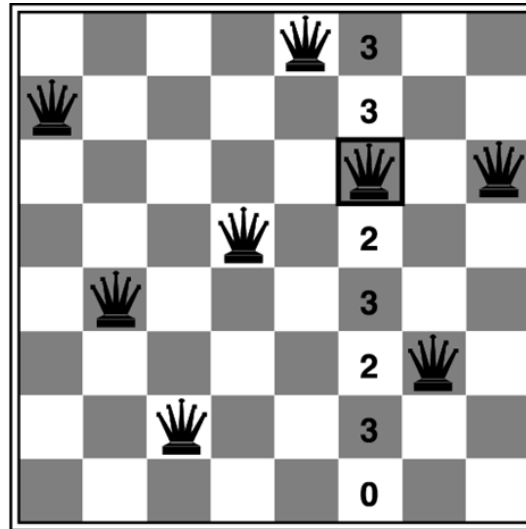
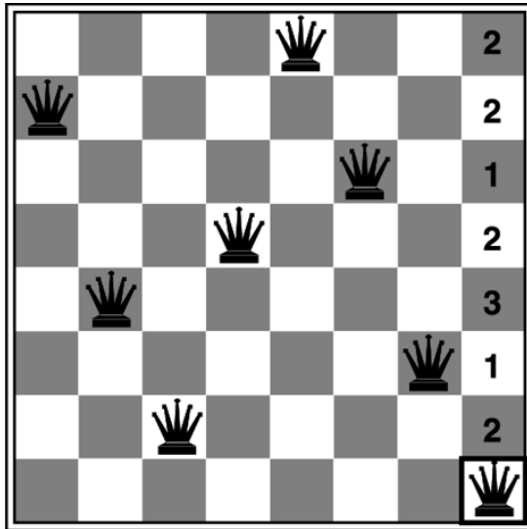
value $\leftarrow$ the value *v* for *var* that minimizes

CONFLICTS(*var*, *v*, *current*, *csp*)

 set *var* = *value* in *current*

return *failure*

Min-Conflicts Example: n-Queens



■ Characteristics

- very effective (though not complete)
 - 1,000,000,000 queens in 50 steps!
 - Hubble observation tasks-scheduling: 3 weeks-> 10 minutes
- suitable for “online” problems!

Related Bibliography

- **Artificial Intelligence: A Modern Approach,**
Stuart Russell and Peter Norvig,
4th edition, Prentice Hall, 2020
 - Sections 5.1 – 5.2

