Constraint Satisfaction Problems (CSPs)

- In Uninformed/Informed Search the objective is:
  - Planning: sequences of actions
    - The path to the goal is the important thing
    - Paths have various costs, depths
    - Heuristics give problem-specific guidance
- CSP is a different (special) kind of search problem:
  - Identification: assignments to variables
    - The goal itself is important, not the path
    - All paths at the same depth (for some formulations)
    - CSPs are specialized for identification problems

Identification in essence is
"Finding/Discovering the Goal"

- **Standard search problems:** (Uniformed/Informed Search)
  - State is a "black box": arbitrary data structure
  - Goal test can be any function over states
  - Successor function can also be anything

- **Constraint satisfaction problems (CSPs):**
  - A special subset of search problems
  - State is defined by variables X_i with values from a domain D (sometimes D depends on i)
  - Goal test is a set of constraints specifying allowable combinations of values for subsets of variables

- Simple example of a *formal representation language*

- Allows useful general-purpose algorithms with more power than standard search algorithms

Example: Map Coloring

- Variables: WA, NT, Q, NSW, V, SA, T
- Domains: $D = \{red, green, blue\}$
- Constraints: adjacent regions must have different colors

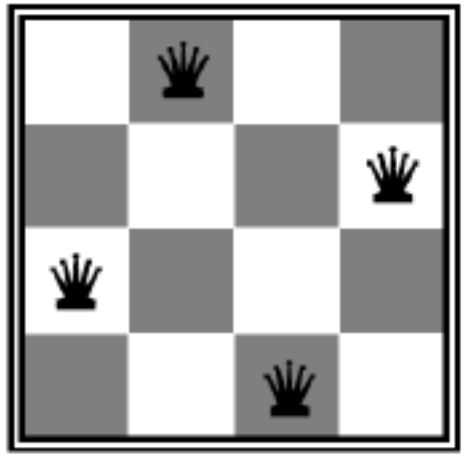
  Implicit: $WA \neq NT$

  Explicit: $(WA, NT) \in \{(red, green), (red, blue), \ldots\}$
- Solutions are assignments satisfying all constraints, e.g.:

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

Example: N-Queens

- Formulation 1:
  - Variables: X_{ij}
  - Domains: $\{0, 1\}$
  - Constraints

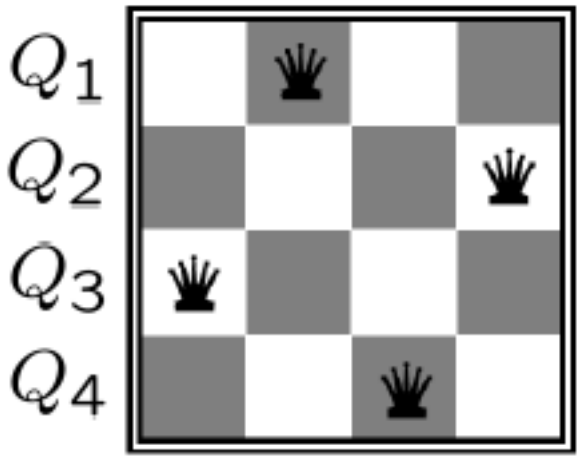


$$\forall i, j, k \quad (X_{ij}, X_{ik}) \in \{(0,0), (0,1), (1,0)\}$$
$$\forall i, j, k \quad (X_{ij}, X_{kj}) \in \{(0,0), (0,1), (1,0)\}$$
$$\forall i, j, k \quad (X_{ij}, X_{i+k,j+k}) \in \{(0,0), (0,1), (1,0)\}$$
$$\forall i, j, k \quad (X_{ij}, X_{i+k,j-k}) \in \{(0,0), (0,1), (1,0)\}$$

$$\sum_{i,j} X_{ij} = N$$

Example: N-Queens

- Formulation 2:
  - Variables: Q_k
  - Domains: $\{1, 2, 3, \ldots N\}$
  - Constraints:



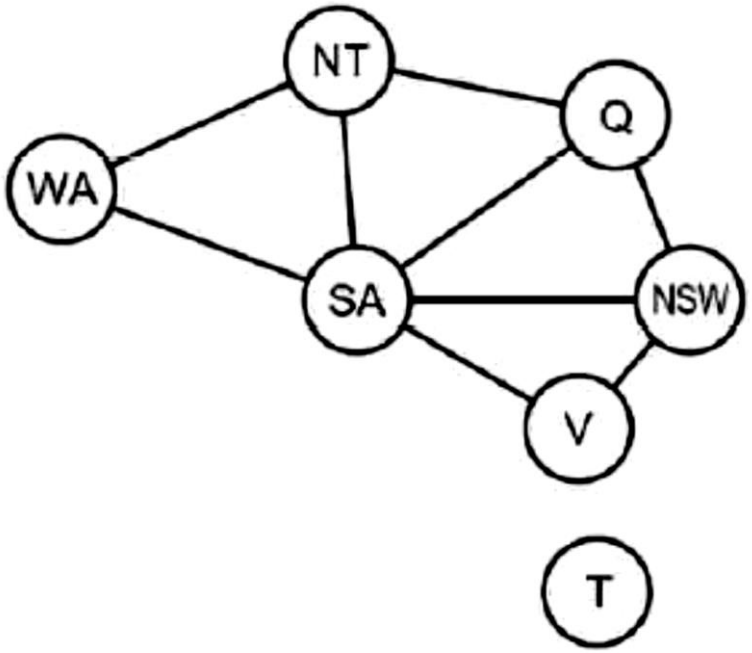
Implicit: $\forall i, j$ non-threatening(Q_i, Q_j)

Explicit: $(Q_1, Q_2) \in \{(1,3), (1,4), \ldots\}$

$\ldots$

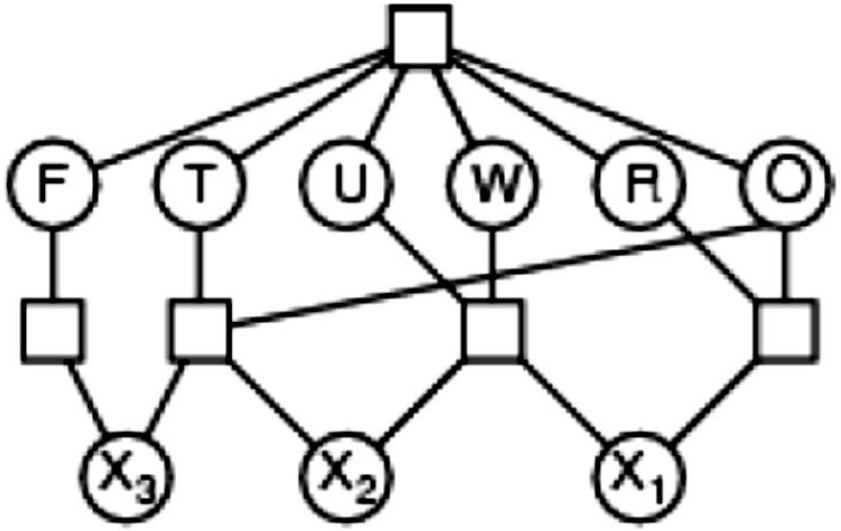
Constraint Graphs

- Binary CSP: each constraint relates (at most) two variables
- Binary constraint graph: nodes are variables, arcs show constraints
- General-purpose CSP algorithms use the graph structure to speed up search. E.g., Tasmania is an independent subproblem!



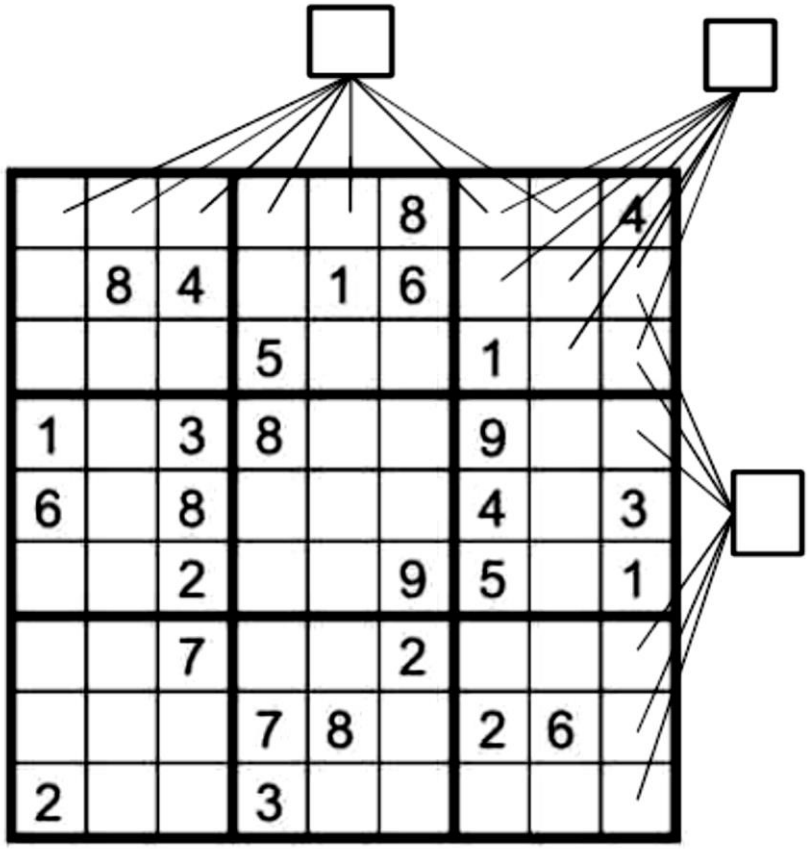
Cryptarithmetic

```
   T W O
+  T W O
-------
F O U R
```



- Variables: $F\ T\ U\ W\ R\ O\ X_1\ X_2\ X_3$
- Domains: $\{0,1,2,3,4,5,6,7,8,9\}$
- Constraints: *Alldiff* (F,T,U,W,R,O)
  - $O + O = R + 10 \cdot X_1$
  - $X_1 + W + W = U + 10 \cdot X_2$
  - $X_2 + T + T = O + 10 \cdot X_3$
  - $X_3 = F,\ T \neq 0,\ F \neq 0$

Sudoku



- Variables:
  - Each (open) square
- Domains:
  - {1,2,…,9}
- Constraints:

  9-way alldiff for each column

  9-way alldiff for each row

  9-way alldiff for each region

  (or can have a bunch of pairwise inequality constraints)

Solution: a value for each cell satisfying the constraints

Scheduling

- Want to schedule a time and a space for each final exam so that no student is scheduled to take more than one final at the same time.
- The space allocated has to be available at the time set.
- The space has to be large enough to accommodate all of the students taking the exam.

Varieties of CSPs

- Discrete Variables
  - **Finite domains**
    - Size d means $O(d^n)$ complete assignments
    - E.g., Boolean CSPs, including Boolean satisfiability (NP-complete)
  - **Infinite domains (integers, strings, etc.)**
    - E.g., job scheduling, variables are start/end times for each job
    - Linear constraints solvable, nonlinear undecidable

- Continuous variables
  - **Linear constraints solvable in polynomial time by LP methods**

Varieties of Constraints

- Varieties of Constraints
  - Unary constraints involve a single variable (equivalent to reducing domains), 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., cryptarithmetic column constraints

- Preferences (soft constraints):
  - E.g., red is better than green
  - Often representable by a cost for each variable assignment
  - Gives constrained optimization problems

Real-World CSPs

- Assignment problems: e.g., who teaches what class
- Timetabling problems: e.g., which class is offered when and where?
- Hardware configuration
- Transportation scheduling
- Factory scheduling
- Circuit layout
- Fault diagnosis
- … lots more!

- Many real-world problems involve real-valued variables…

Solving CSPs

- Standard search formulation of CSPs

- States defined by the values assigned so far (partial assignments)
  - Initial state: the empty assignment, {}
  - Successor function: assign a value to an unassigned variable
  - Goal test: the current assignment is complete and satisfies all constraints

Solving CSPs (continued)

- A *state* is an *assignment* of values to some or all variables.
  - An assignment is *complete* when every variable has a value.
  - An assignment is partial when some variables have no values.
- ***Consistent assignment***
  - assignment does not violate the constraints
- A ***solution*** to a CSP is a complete and consistent assignment.
- Some CSPs require a solution that maximizes an *objective function*.

Backtracking Search

- Backtracking search is the basic uninformed algorithm for solving CSPs
- Idea 1: One variable at a time
  - Variable assignments are commutative, so fix ordering
  - I.e., [WA = red then NT = green] same as [NT = green then WA = red]
  - Only need to consider assignments to a single variable at each step
- Idea 2: Check constraints as you go
  - I.e. consider only values which do not conflict previous assignments
  - Might have to do some computation to check the constraints
  - "Incremental goal test"
- Depth-first search with these two improvements is called *backtracking search* (not the best name)
- Can solve n-queens for $n \approx 25$

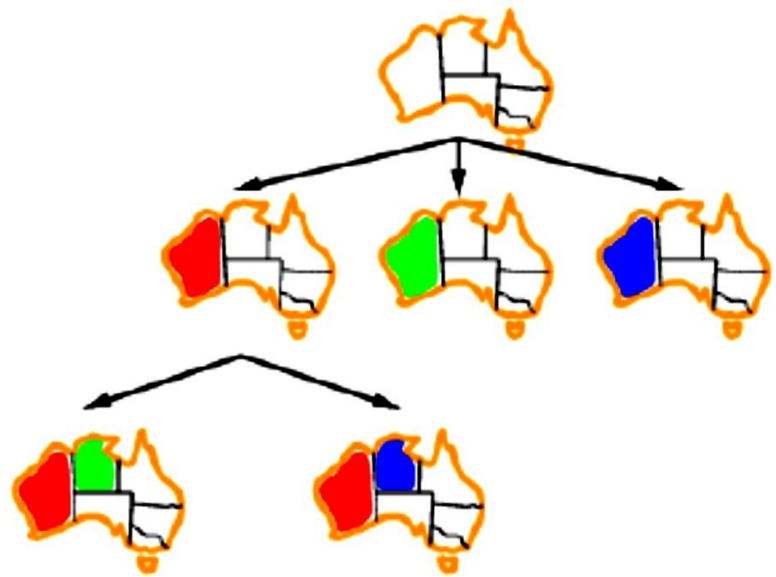
Backtracking Search – Pseudo-Code

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

function RECURSIVE-BACKTRACKING(assignment, csp) returns a solution, or
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 according to Constraints[csp] then
            add { var = value } to assignment
            result ← RECURSIVE-BACKTRACKING(assignment, csp)
            if result ≠ failue then return result
            remove { var = value } from assignment
    return failure
```

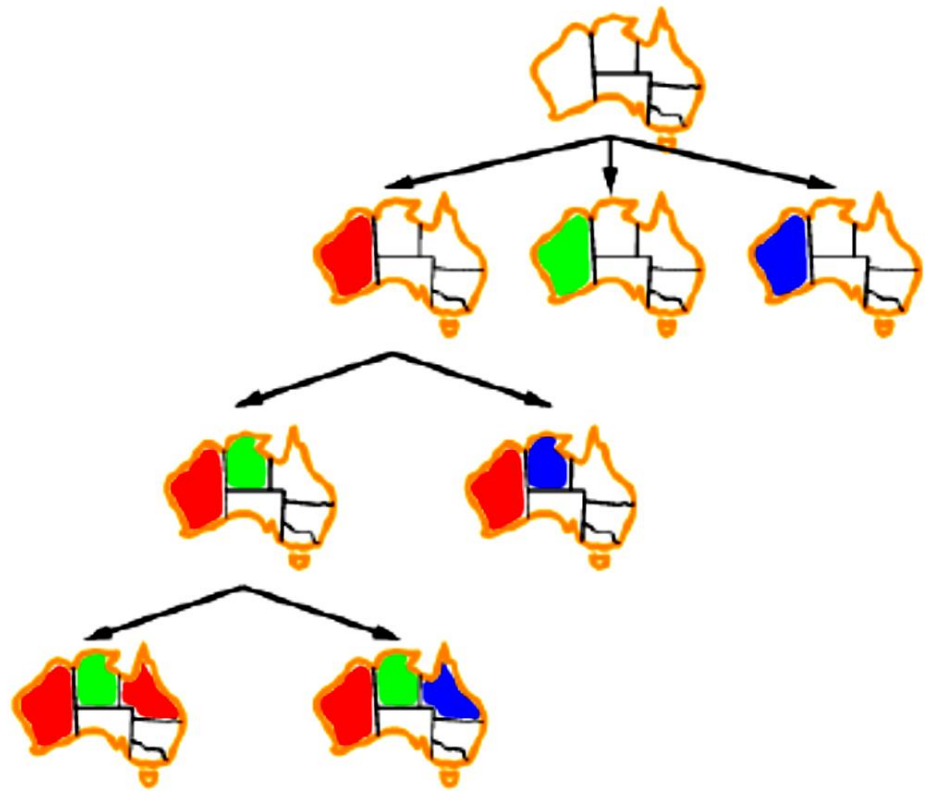
Backtracking example

Backtracking example

Backtracking example



Backtracking example

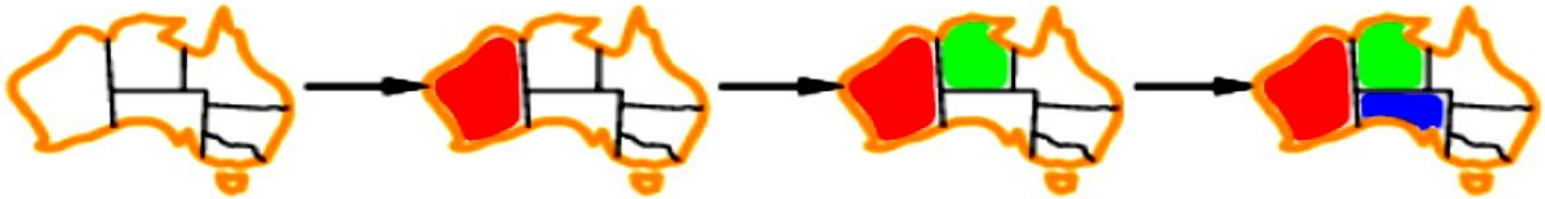


Improving Backtracking

- General-purpose ideas give huge gains in speed
- Ordering:
  - Which variable should be assigned next?
  - In what order should its values be tried?
- Filtering: Can we detect inevitable failure early?
- Structure: Can we exploit the problem structure?

Most constrained variable

- Most constrained variable:
  choose the variable with the fewest legal values

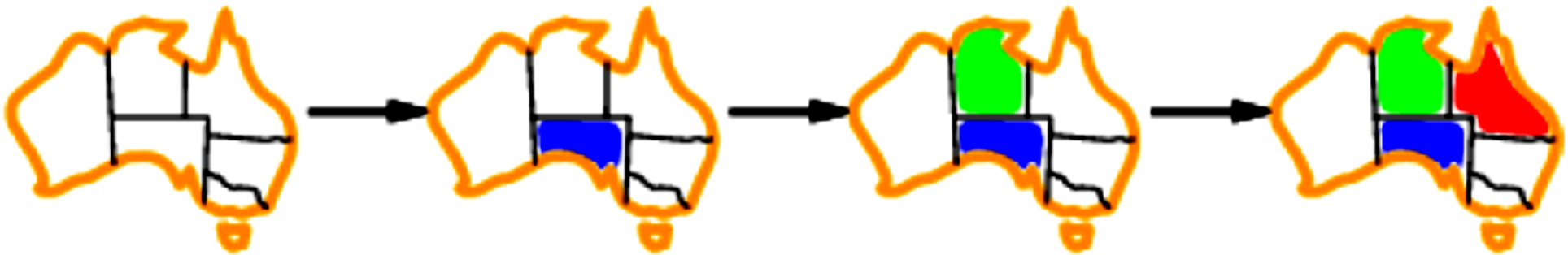


- a.k.a. minimum remaining values (MRV) heuristic

- Fail-first heuristic
  If a variable has no legal values that will be picked first and fail

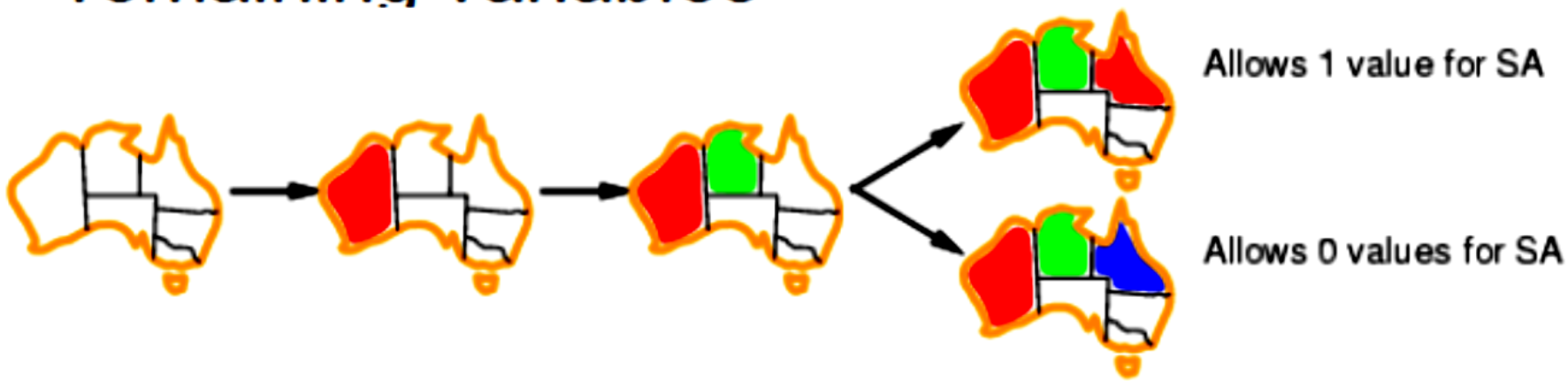
Most constraining variable

- Tie-breaker among most constrained variables
- Most constraining variable:
  - choose the variable with the most constraints on remaining variables



Least constraining value

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



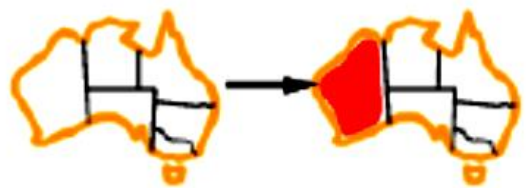
- Combining these heuristics makes 1000 queens feasible

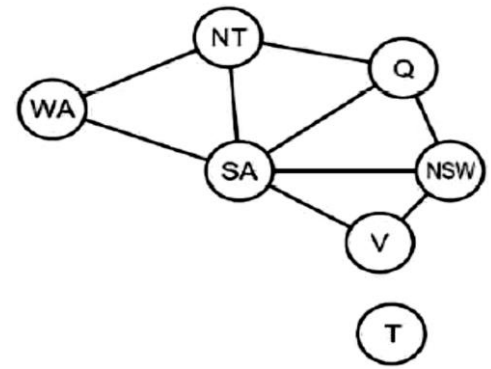
Forward checking

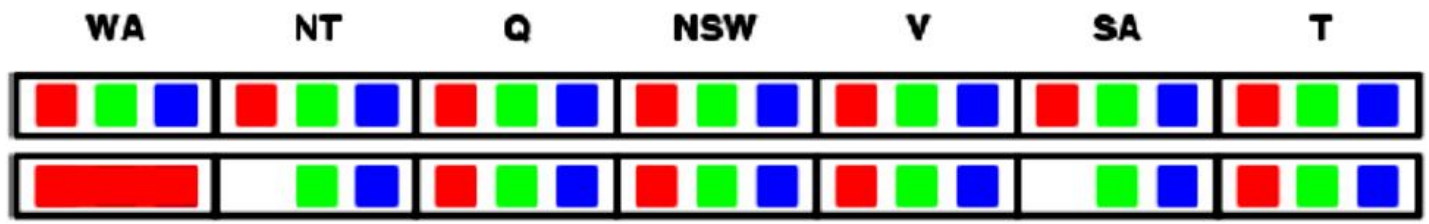
- Idea:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values

Forward checking

- Idea:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values

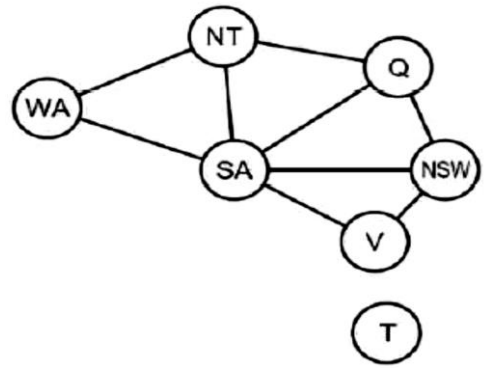






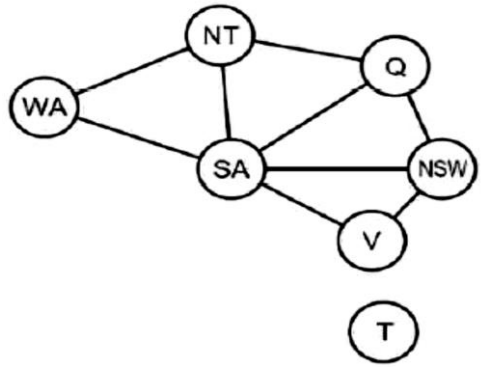
Forward checking

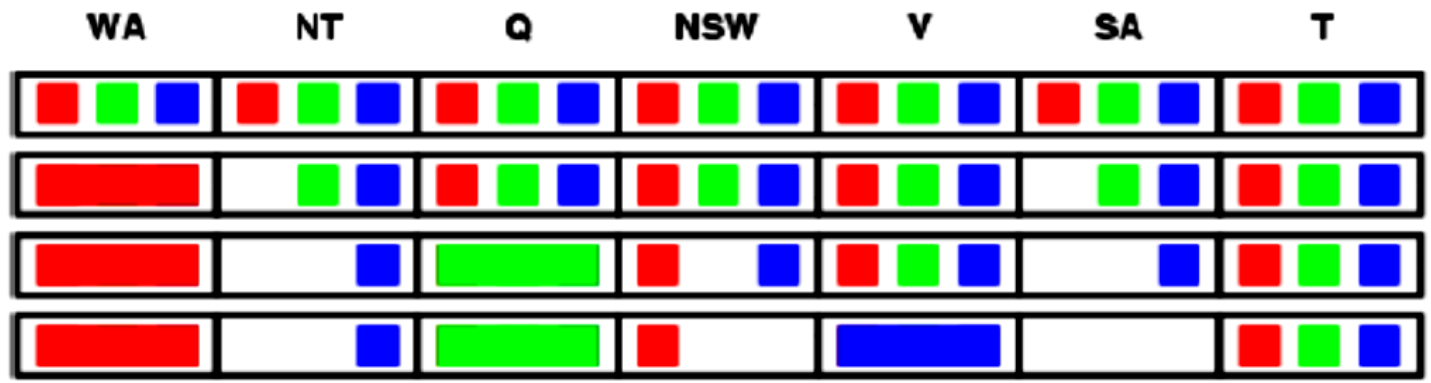
- Idea:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values



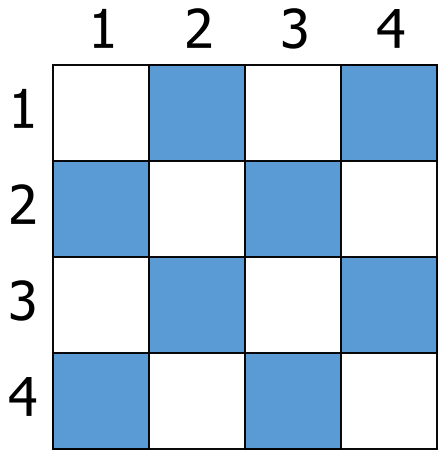
Forward checking

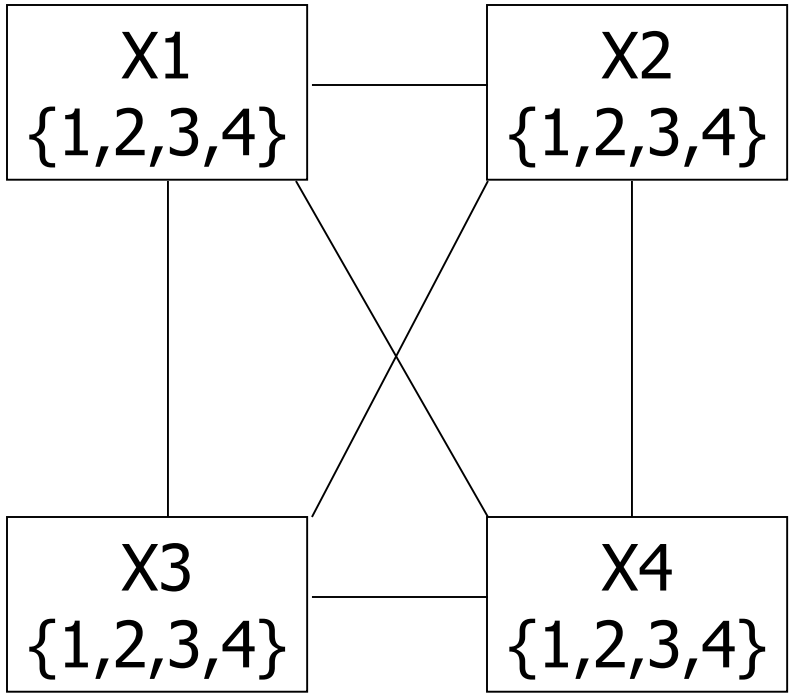
- Idea:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values





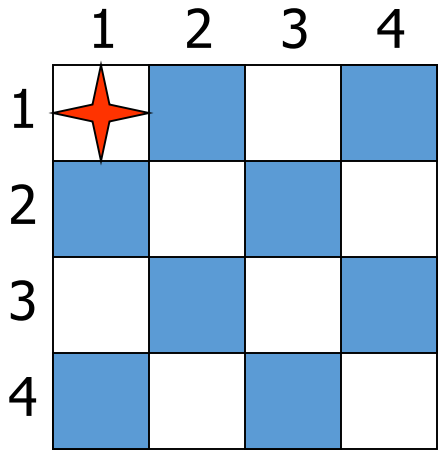
Another Forward Checking Example: 4-Queens Problem

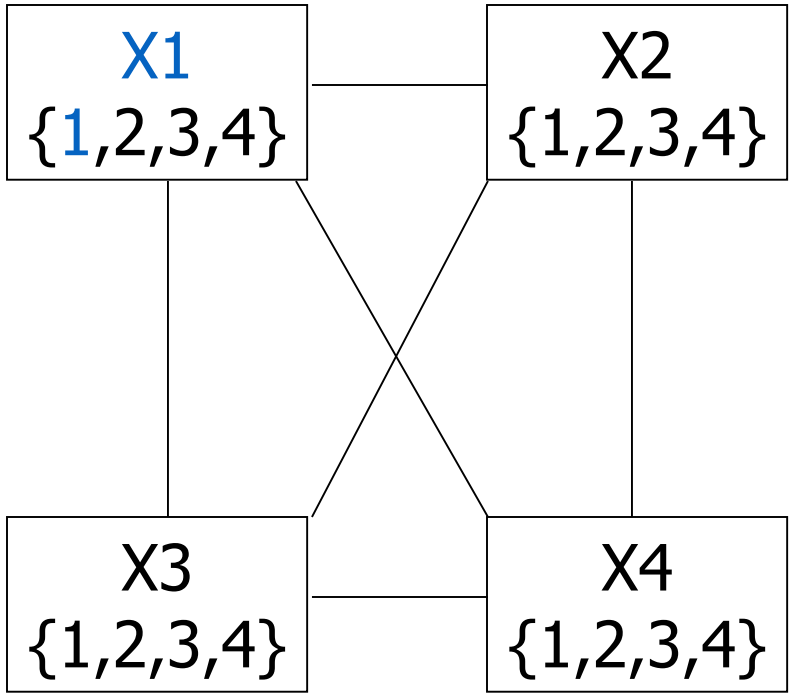




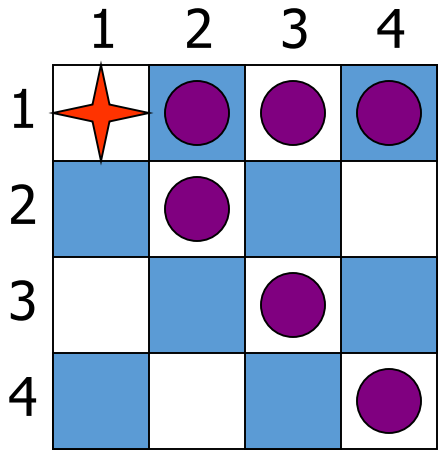
Xi = i^{th} Queen can be placed anywhere in column i

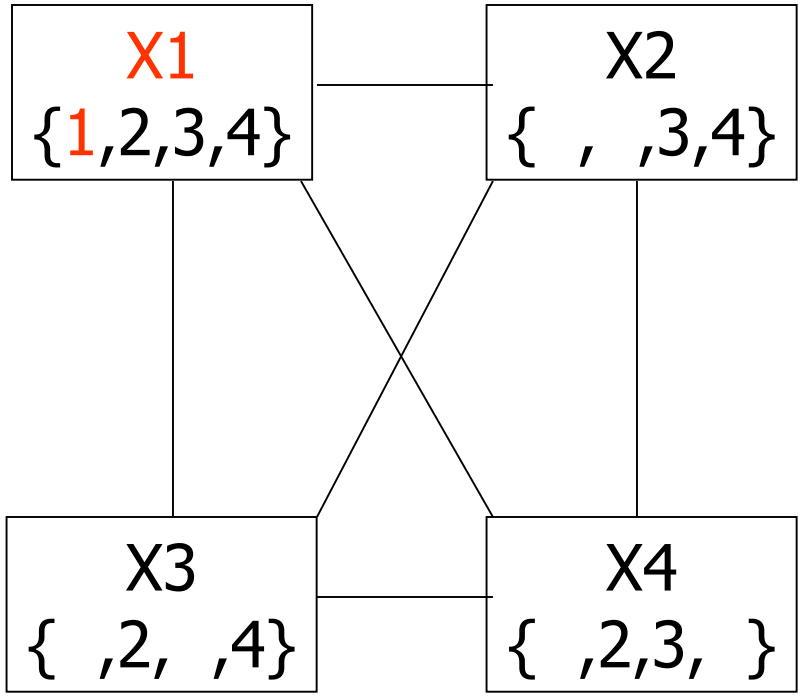
Example: 4-Queens Problem



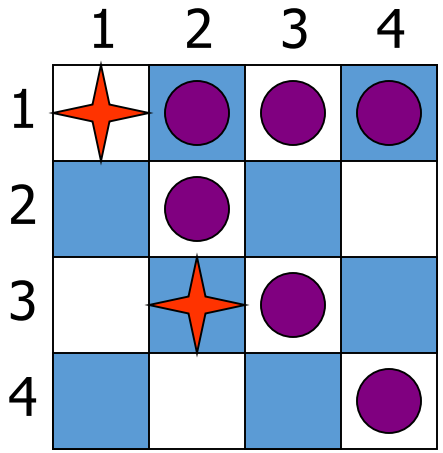


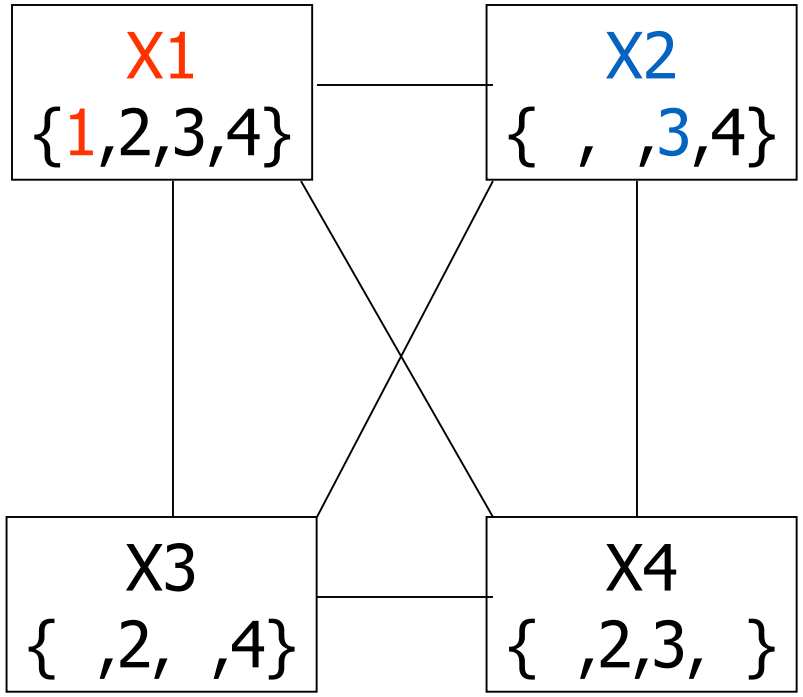
Example: 4-Queens Problem



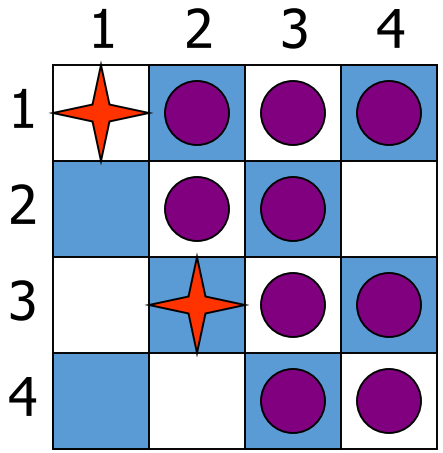


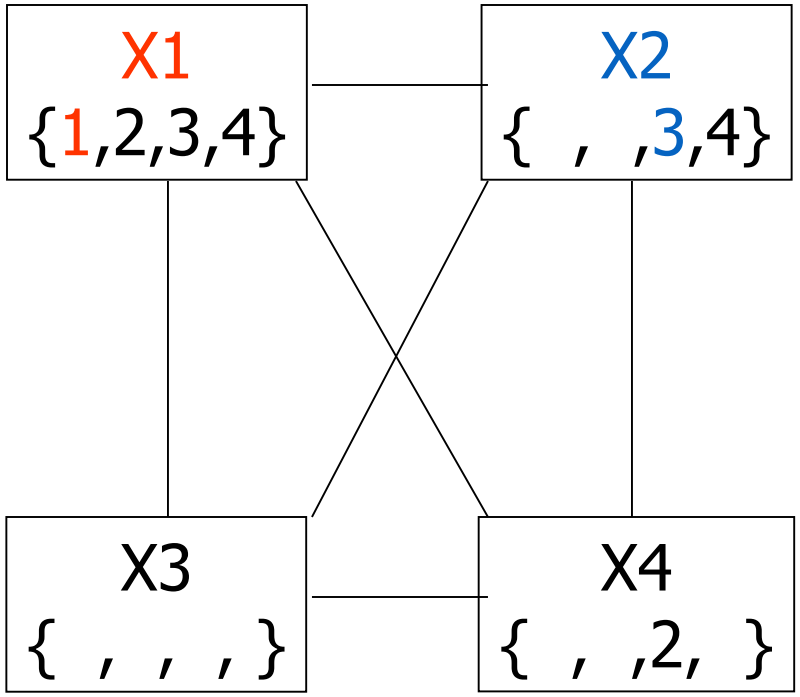
Example: 4-Queens Problem



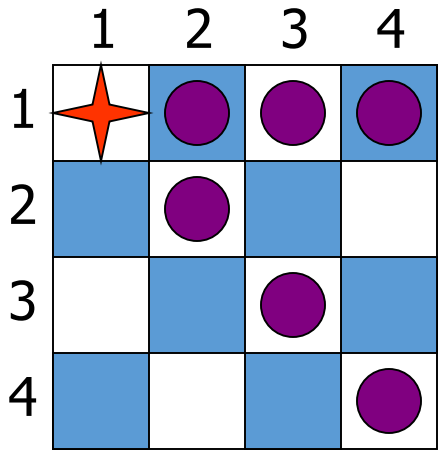


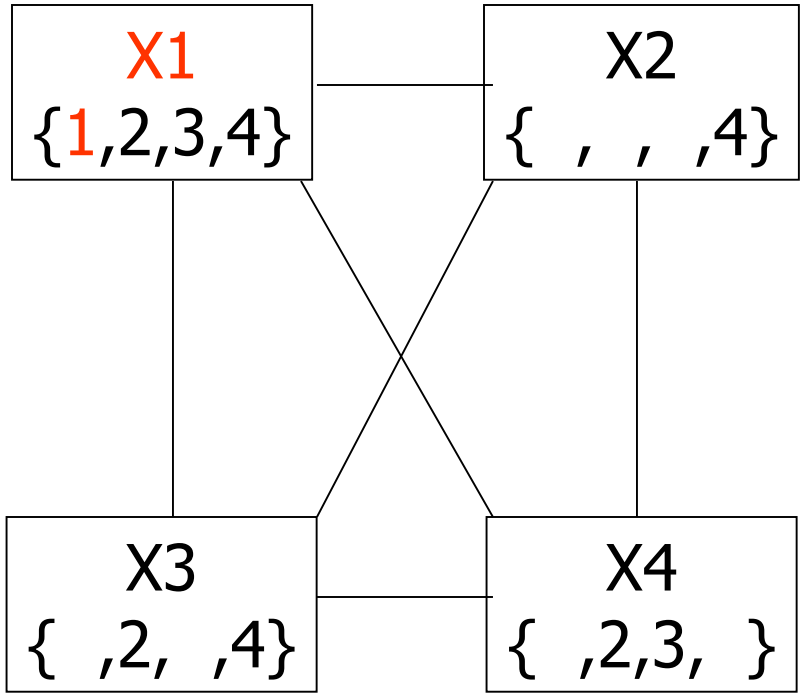
Example: 4-Queens Problem



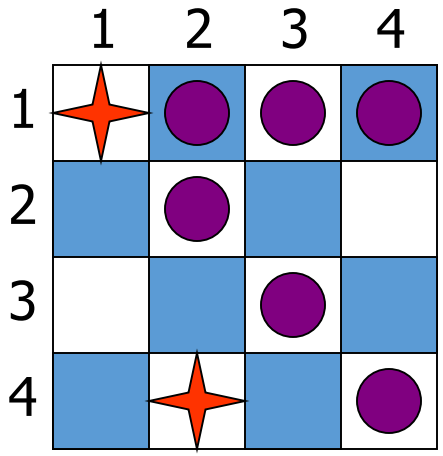


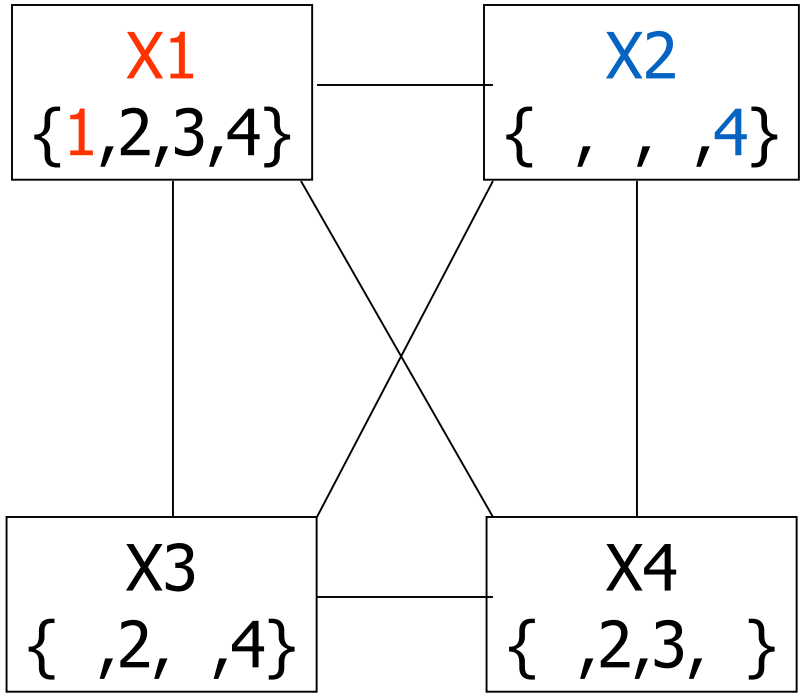
Example: 4-Queens Problem



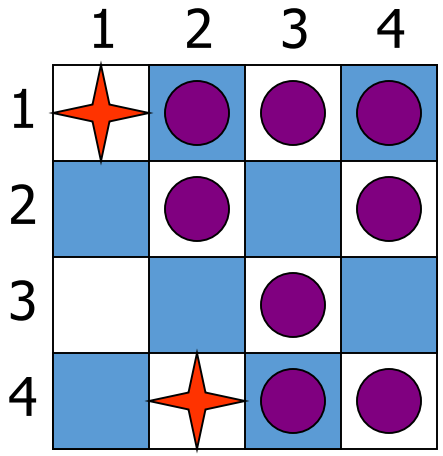


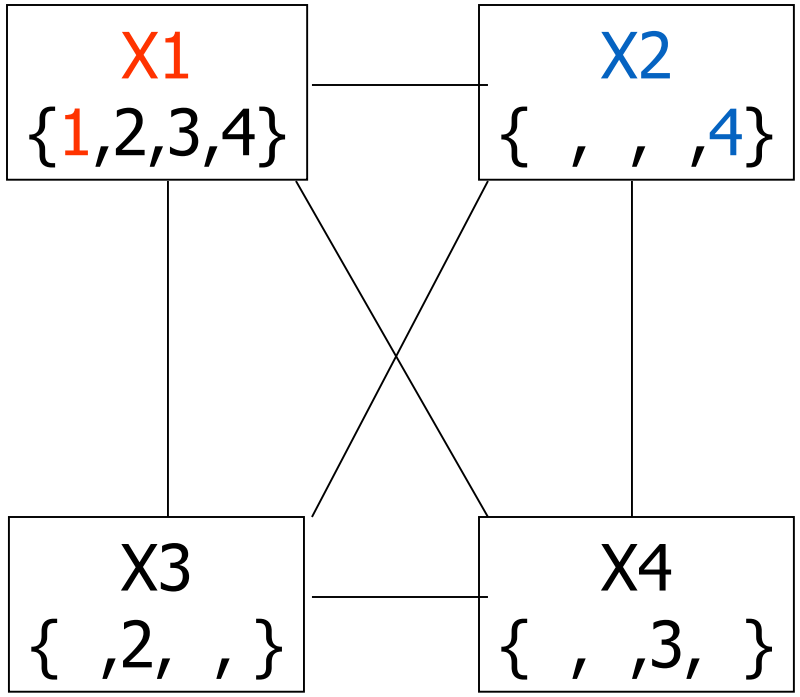
Example: 4-Queens Problem



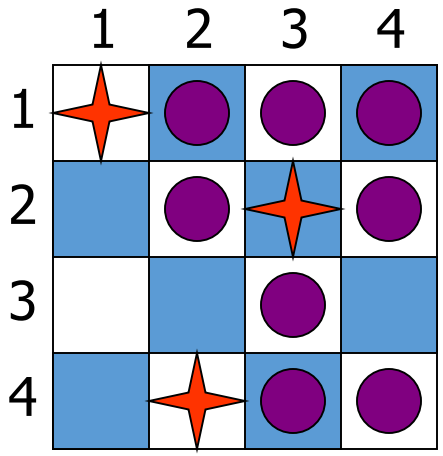


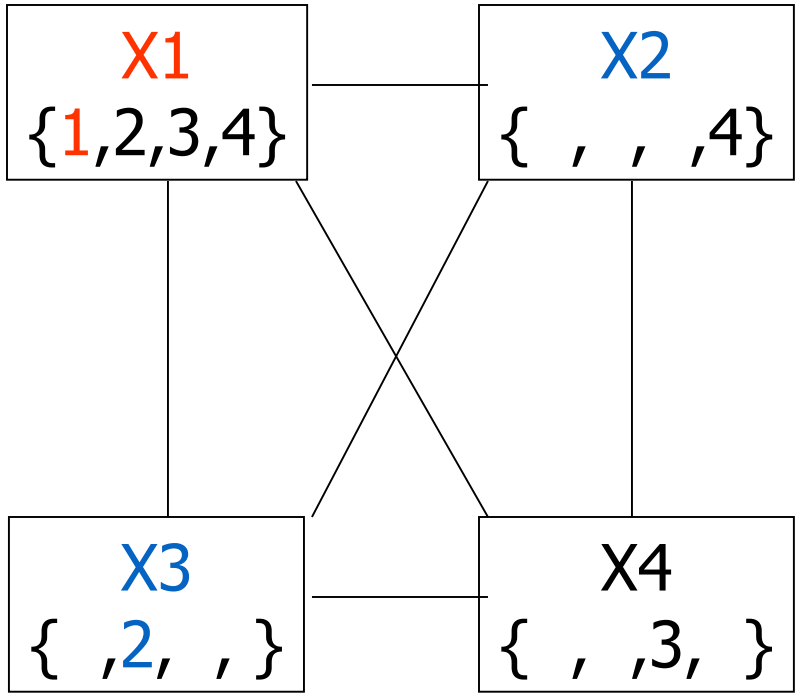
Example: 4-Queens Problem



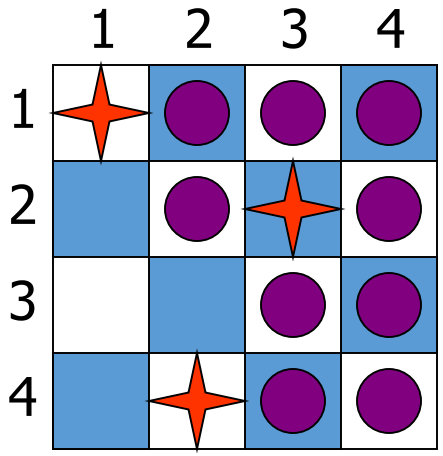


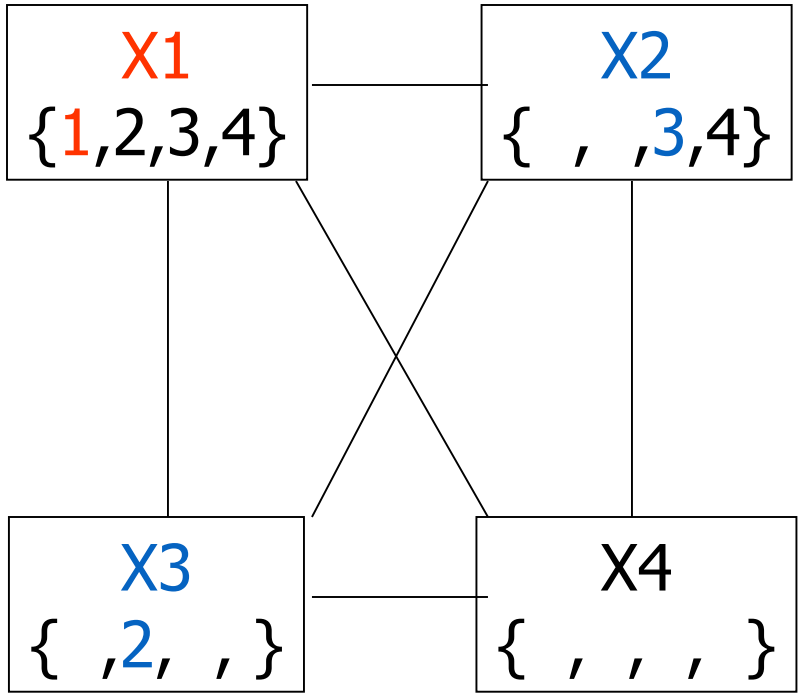
Example: 4-Queens Problem





Example: 4-Queens Problem





Comparison of CSP algorithms on different problems

Problem	Backtracking	BT+MRV	Forward Checking	FC+MRV
USA	(> 1,000K)	(> 1,000K)	2K	60
n-Queens	(> 40,000K)	13,500K	(> 40,000K)	817K
Zebra	3,859K	1K	35K	0.5K
Random 1	415K	3K	26K	2K
Random 2	942K	27K	77K	15K

Median number of consistency checks over 5 runs to solve problem

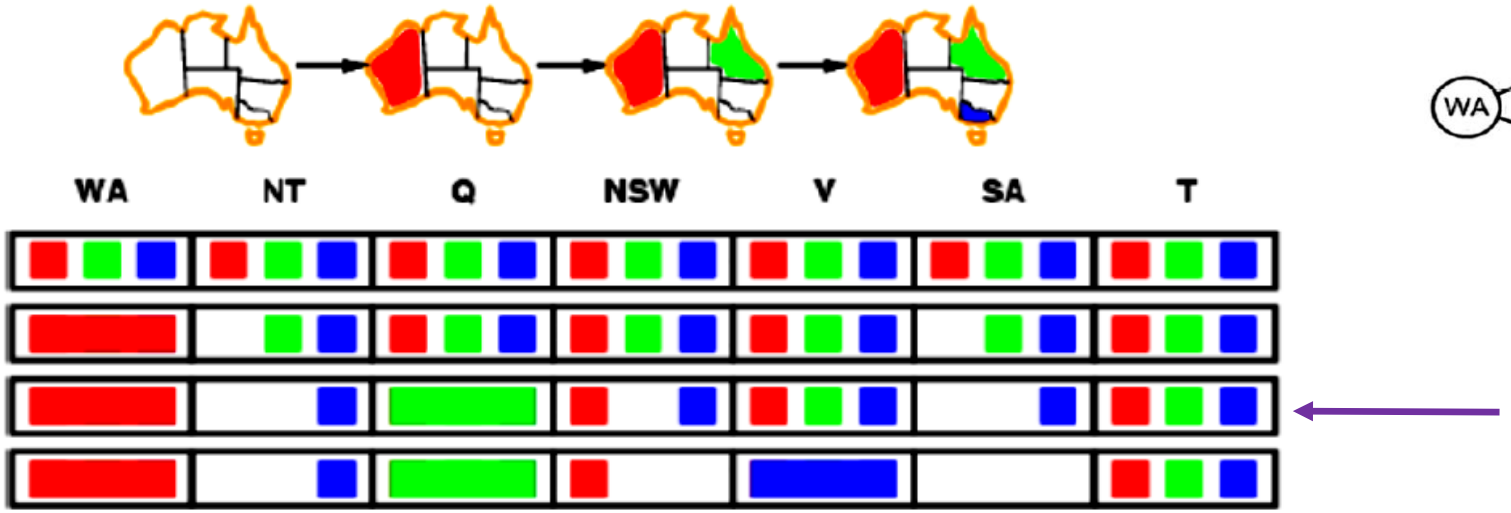
Parentheses -> no solution found

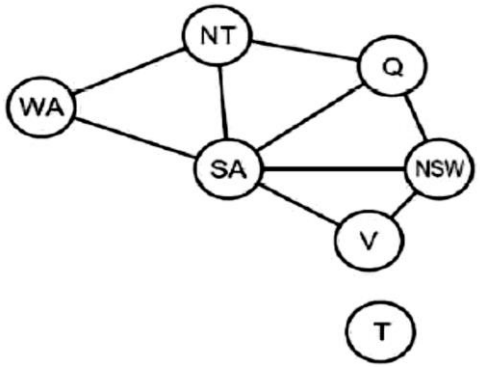
USA: 4 coloring

n-Queens: n ranges from 2 to 50

Constraint propagation

- Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures:



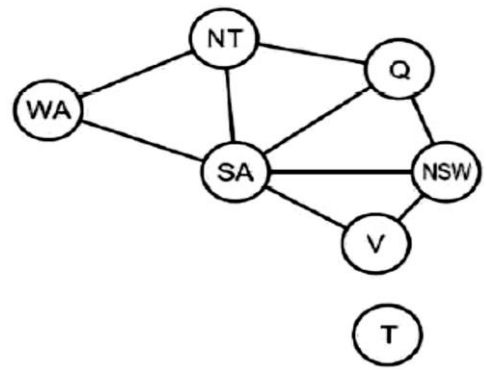


- NT and SA cannot both be blue!
- Constraint propagation repeatedly enforces constraints locally

Arc consistency

- Simplest form of propagation makes each arc consistent
- $X \rightarrow Y$ is consistent iff

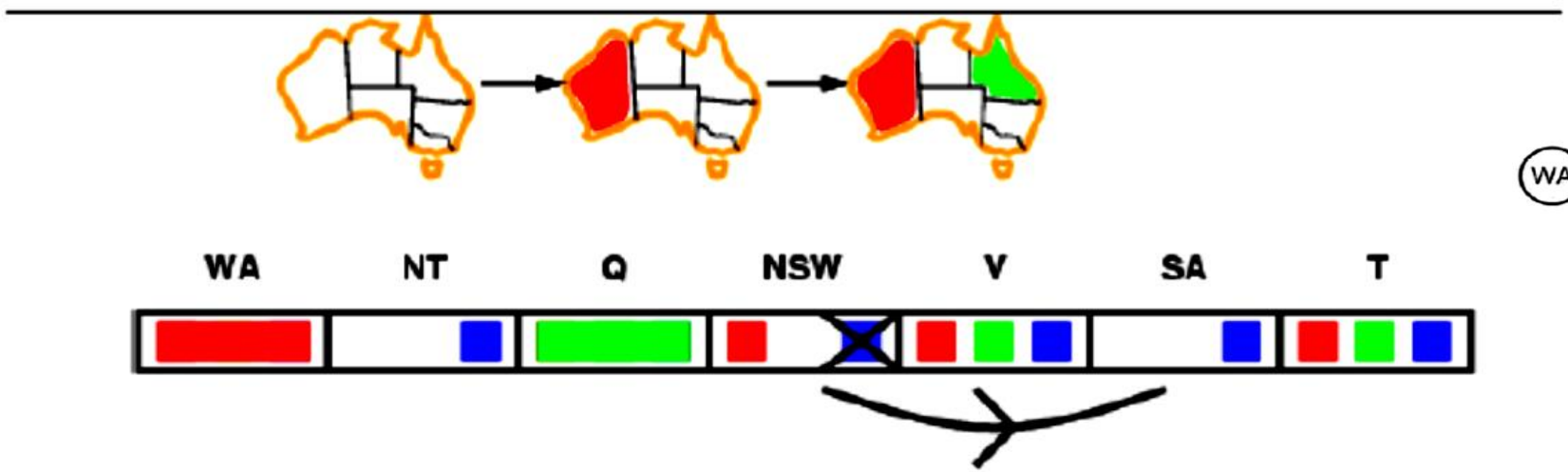
  for every value x of X there is some allowed y

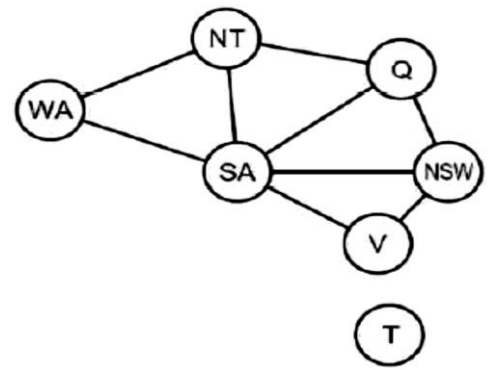


Arc consistency

- 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

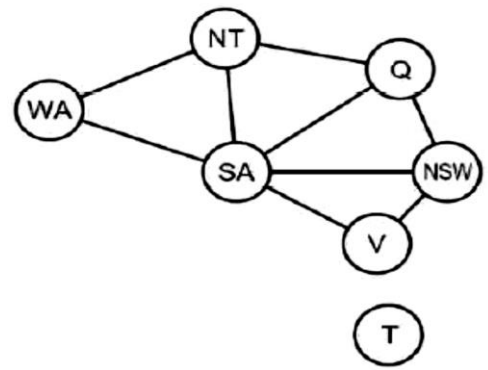


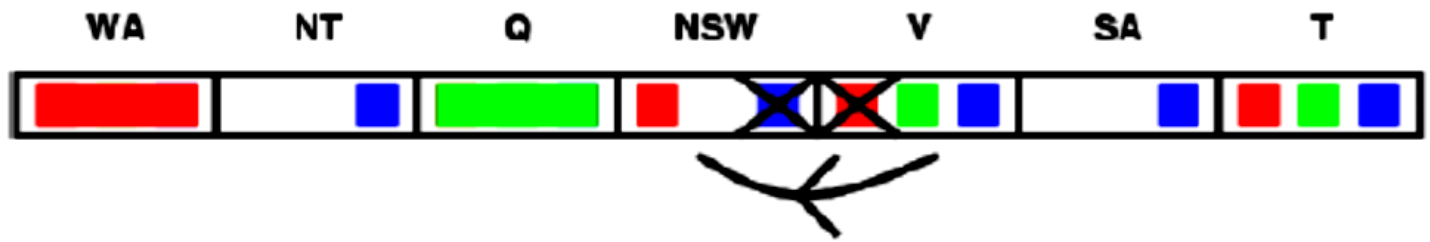


Arc consistency

- 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



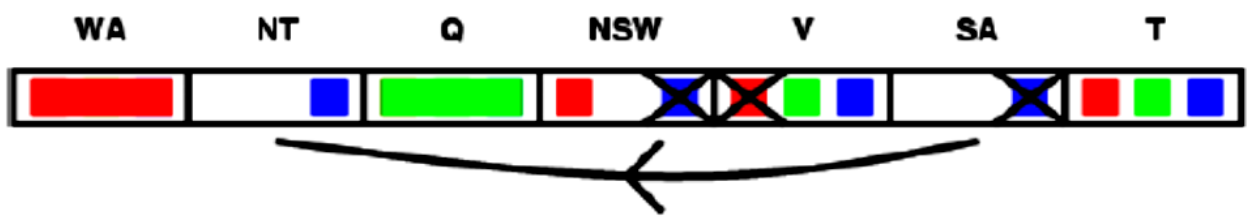


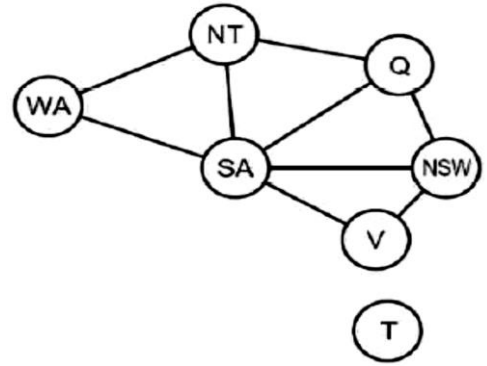
- If X loses a value, neighbors of X need to be rechecked

Arc consistency

- 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



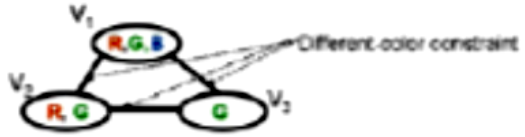


- If X loses a value, neighbors of X need to be rechecked
- Arc consistency detects failure earlier than forward checking
- Can be run as a preprocessor or after each assignment

Constraint Propagation Example 2:

Graph Coloring

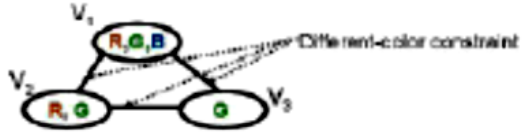
Initial Domains are indicated



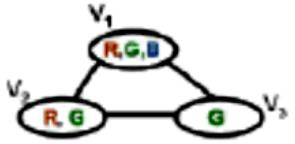
Arc examined	Value deleted

Graph Coloring

Initial Domains are indicated

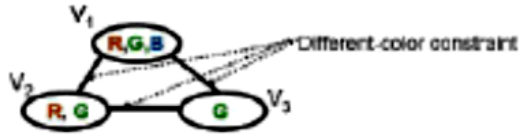


Arc examined	Value deleted
$V_1 - V_2$	none

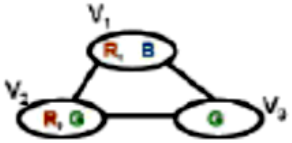


Graph Coloring

Initial Domains are indicated

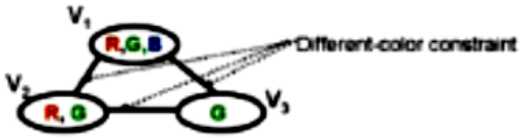


Arc examined	Value deleted
$V_1 - V_2$	none
$V_3 - V_2$	V_1(G)

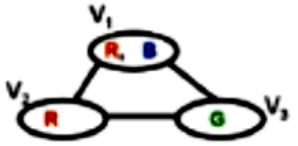


Graph Coloring

Initial Domains are indicated

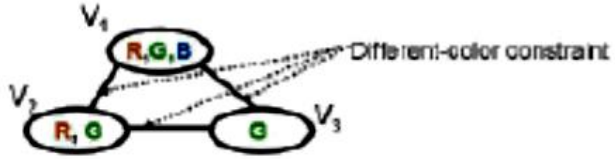


Arc examined	Value deleted
$V_1 - V_2$	none
$V_1 - V_3$	V_1(G)
$V_2 - V_3$	V_2(G)

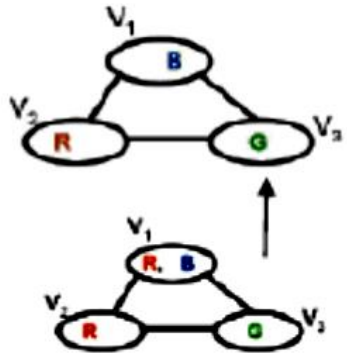


Graph Coloring

Initial Domains are indicated

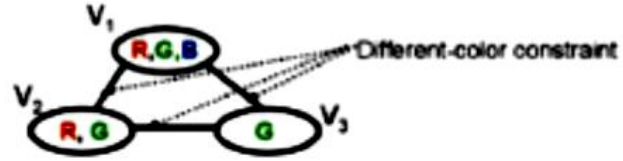


Arc examined	Value deleted
$V_1 - V_2$	none
$V_1 - V_3$	V_1(G)
$V_2 - V_3$	V_2(G)
$V_1 - V_2$	V_1(R)

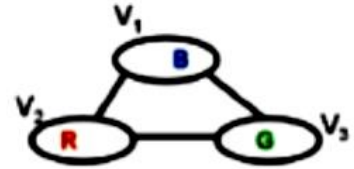


Graph Coloring

Initial Domains are indicated

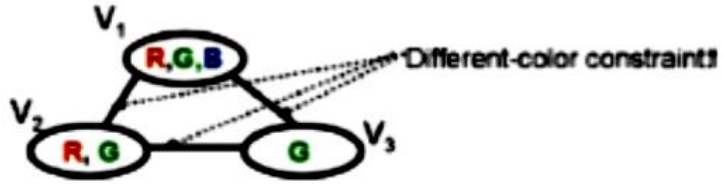


Arc examined	Value deleted
$V_1 - V_2$	none
$V_1 - V_3$	V_1(G)
$V_2 - V_3$	V_2(G)
$V_1 - V_2$	V_1(R)
$V_1 - V_3$	none

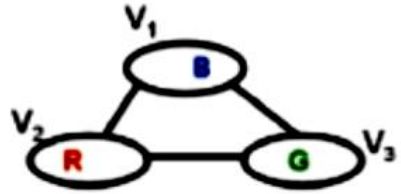


Graph Coloring

Initial Domains are indicated



Arc examined	Value deleted
$V_1 - V_2$	none
$V_1 - V_3$	V_1(G)
$V_2 - V_3$	V_2(G)
$V_1 - V_2$	V_1(R)
$V_1 - V_3$	none
$V_2 - V_3$	none



Arc consistency algorithm AC-3

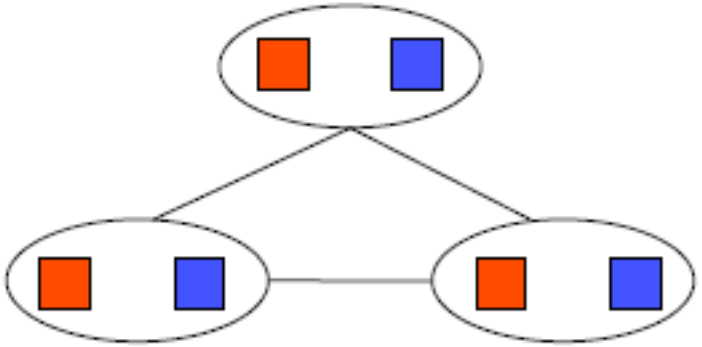
```
function AC-3( csp) returns the CSP, possibly with reduced domains
    inputs: csp, a binary CSP with variables {X_1, X_2, ..., X_n}
    local variables: queue, a queue of arcs, initially all the arcs in csp

    while queue is not empty do
        (X_i, X_j) ← REMOVE-FIRST(queue)
        if RM-INCONSISTENT-VALUES(X_i, X_j) then
            for each X_k in NEIGHBORS[X_i] do
                add (X_k, X_i) to queue

---

function RM-INCONSISTENT-VALUES( X_i, X_j) returns true iff remove a value
    removed ← false
    for each x in DOMAIN[X_i] do
        if no value y in DOMAIN[X_j] allows (x,y) to satisfy constraint(X_i, X_j)
            then delete x from DOMAIN[X_i]; removed ← true
    return removed
```

- Time complexity: $O(n^2d^3)$

The story so far:

- CSPs are a special kind of problem:
  - 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 significantly
- Forward checking prevents assignments that guarantee later failure
- Constraint propagation (e.g., arc consistency) does additional work to constrain values and detect inconsistencies

Limitations of Arc Consistency



Arc consistent but no sloution

K-Consistency

- Increasing degrees of consistency
- 1-Consistency (Node Consistency): Each single node's domain has a value which meets that node's unary constraints
- 2-Consistency (Arc Consistency): For each pair of nodes, any consistent assignment to one can be extended to the other
- K-Consistency: For each k nodes, any consistent assignment to k-1 can be extended to the k^{th} node.

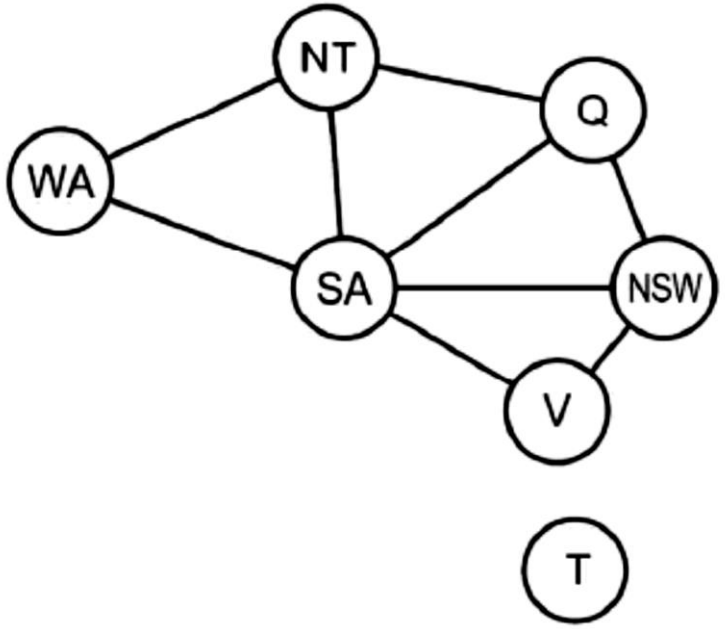
- Higher k more expensive to compute

Trade-offs

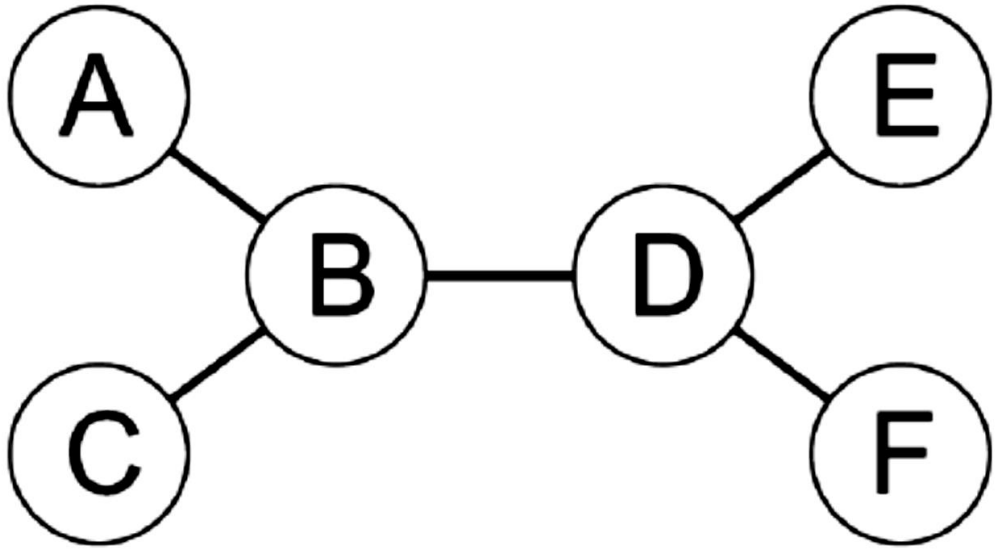
- Running stronger consistency checks…
  - Takes more time
  - But will reduce branching factor and detect more inconsistent partial assignments
  - No “free lunch”
    - In worst case n-consistency takes exponential time
- Generally helpful to enforce 2-Consistency (Arc Consistency)
- Sometimes helpful to enforce 3-Consistency
- Higher levels may take more time to enforce than they save.

Exploiting Graph Structure

- Extreme case: independent subproblems
  - Example: Tasmania and mainland do not interact
- Independent subproblems are identifiable as connected components of constraint graph
- Suppose a graph of n variables can be broken into sub-problems of only c variables: (assume c constant)
  - Worst-case solution cost is $O((n/c)(d^c))$, linear in n
  - E.g., n = 80, d = 2, c =20
  - 2^{80} = 4 billion years at 10 million nodes/sec
  - $(4)(2^{20})$ = 0.4 seconds at 10 million nodes/sec



Tree-Structured CSPs

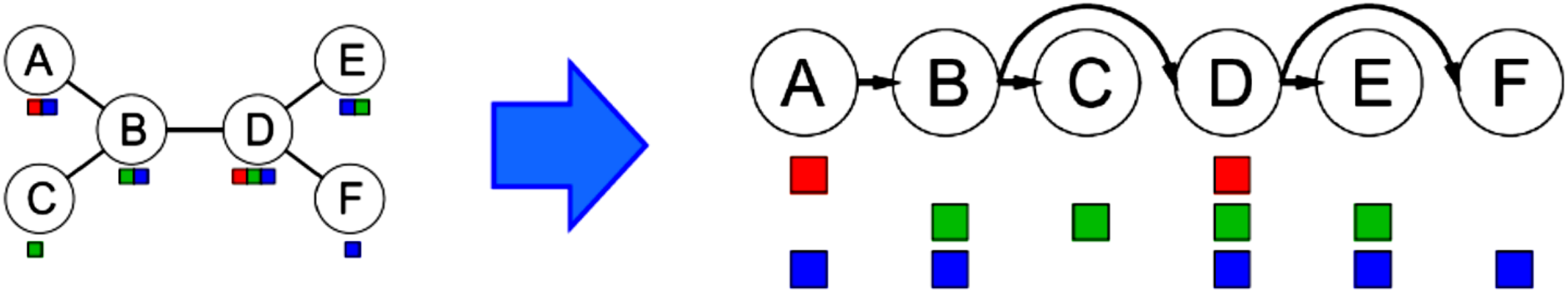


- Theorem: if the constraint graph has no loops, the CSP can be solved in $O(n\ d^2)$ time
  - Compare to general CSPs, where worst-case time is $O(d^n)$

This is linear in n, the number of variables!

Algorithm for Tree-Structured CSPs

- **Algorithm for tree-structured CSPs:**
  - Order: Choose a root variable, order variables so that parents precede children



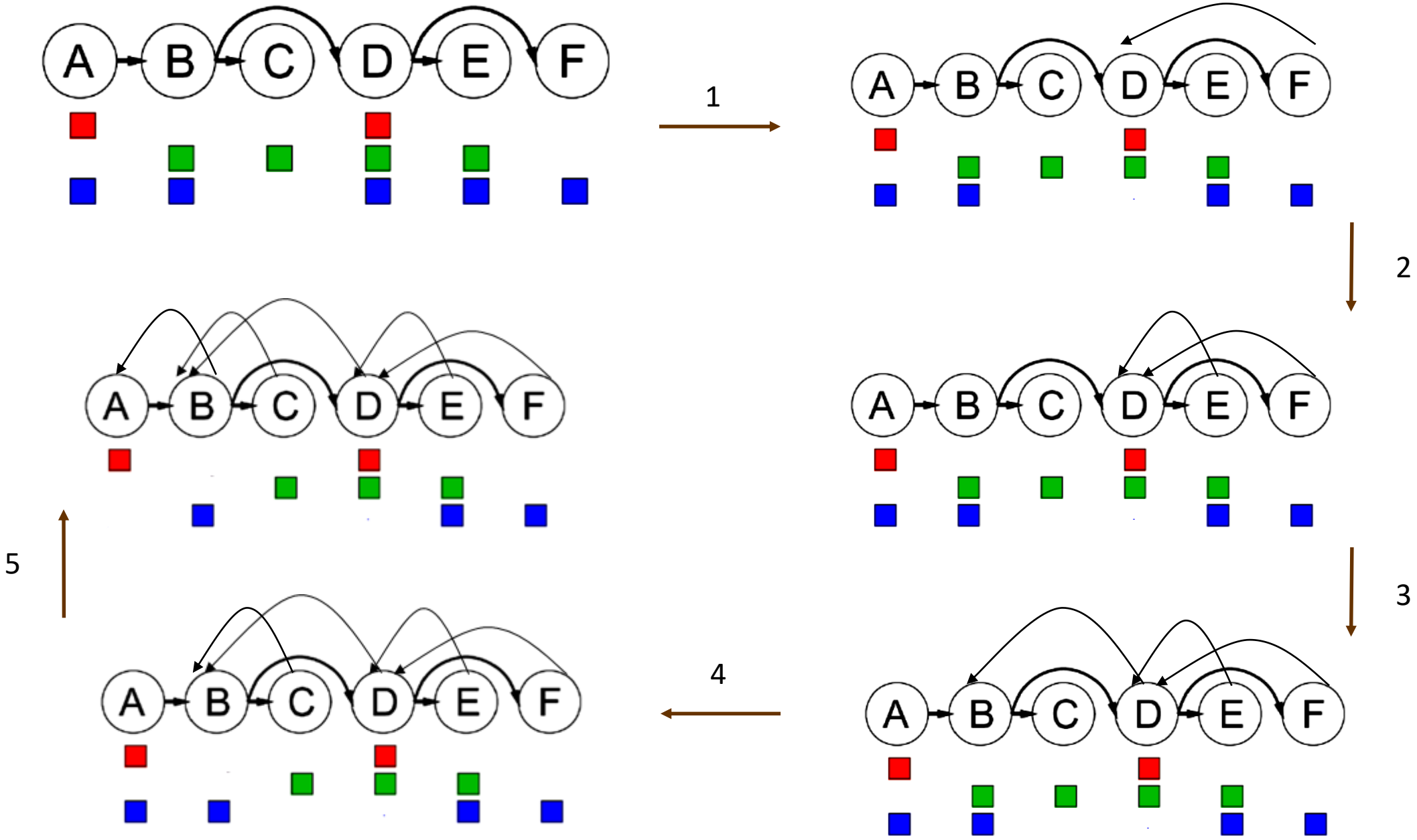
– Backward Pass

For j from n down to 2, apply arc consistency to arc [Parent(X_j), X_j)]
Remove values from Parent(X_j) if needed

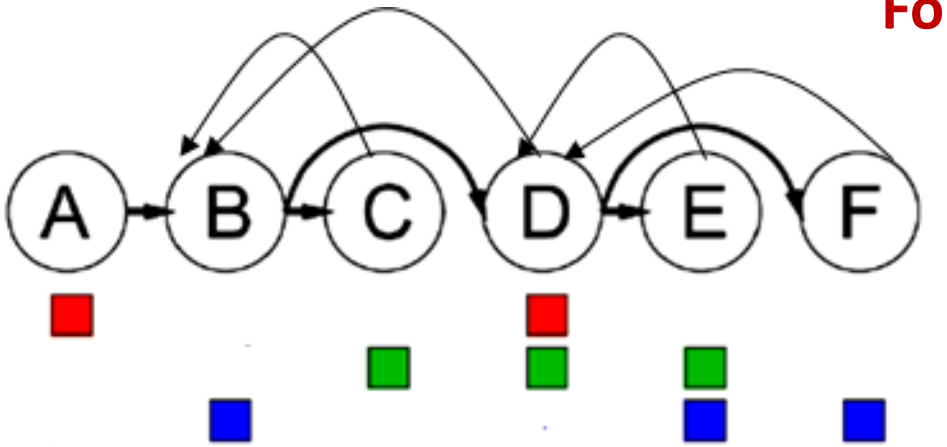
– Forward Pass

For j from 1 to n assign X_j consistently with Parent(X_j)

Backward Pass in Action

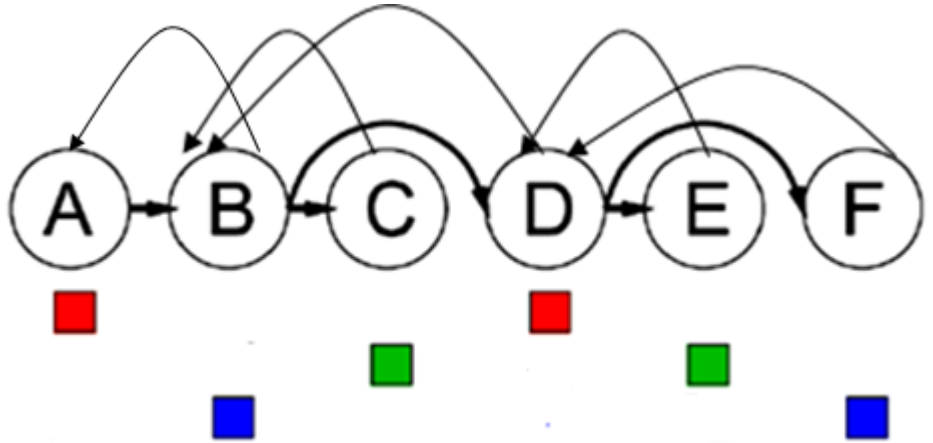


Forward Pass in Action: (3 solutions)

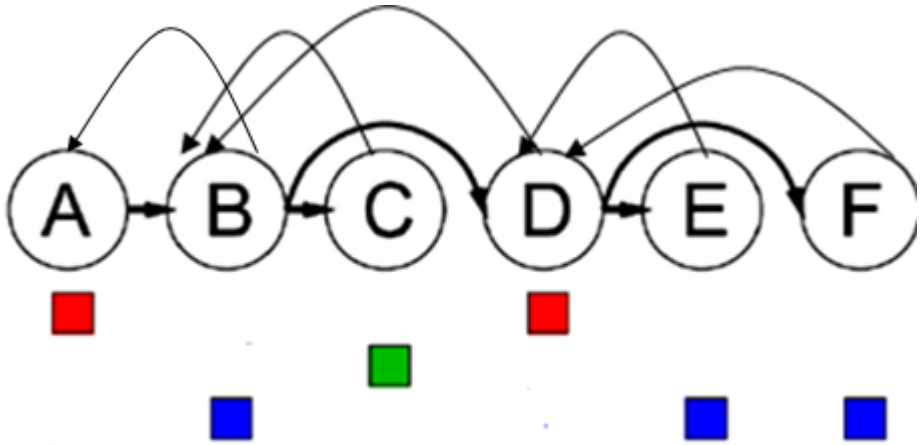


Result of Backward Pass

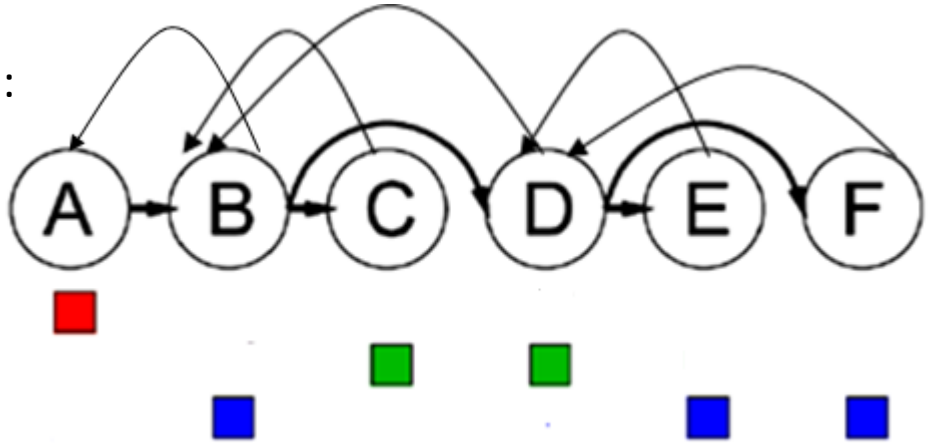
1:



2:

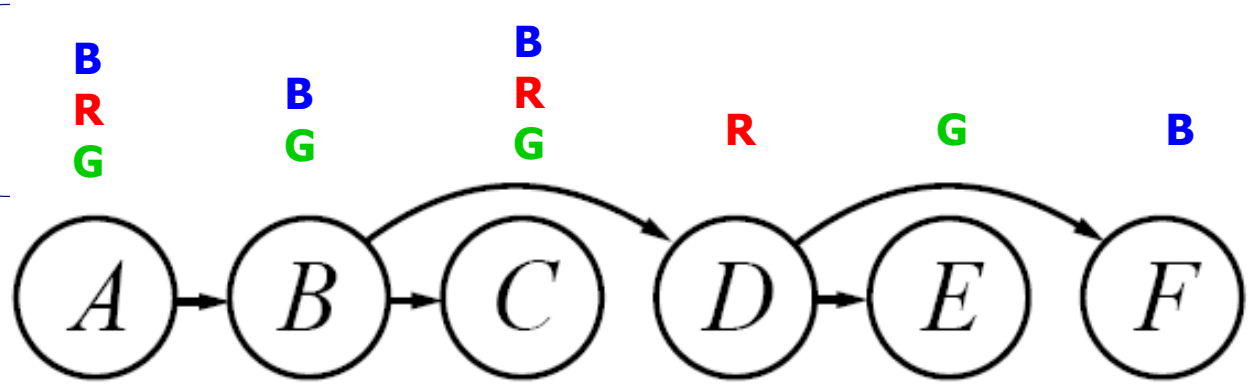


3:



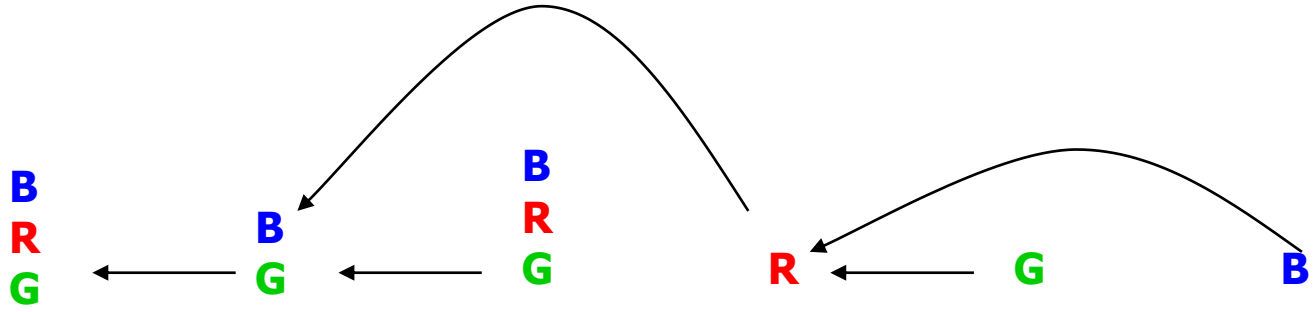
Tree CSP in Action

Initial Values on Variables

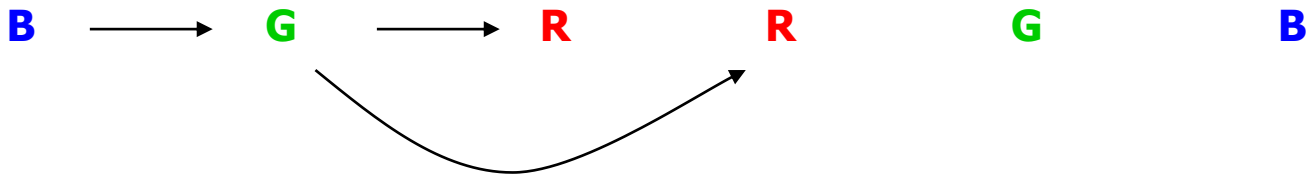


Tree CSP in Action:

Backward Pass
(constraint
propagation)



Forward Pass
(assignment)



Why does it work?

- Claim 1: After backward pass, all root-to-leaf arcs are consistent
- Proof: Each X→Y was made consistent at one point and Y's domain could not have been reduced thereafter (because Y's children were processed before Y)

- Claim 2: If root-to-leaf arcs are consistent, forward assignment will not backtrack
- Proof: Induction on position

- Why doesn't this algorithm work with cycles in the constraint graph?

Tree CSP Complexity

- Backward pass
  - n arc checks
  - Each has complexity d^2 at worst

- Forward pass
  - n variable assignments, O(nd)

  $\Rightarrow$ Overall complexity is $O(nd^2)$

Algorithm works because if the backward pass succeeds, then every variable by definition has a legal assignment in the forward pass

What about non-tree CSPs?

- General idea is to convert the graph to a tree

2 general approaches:

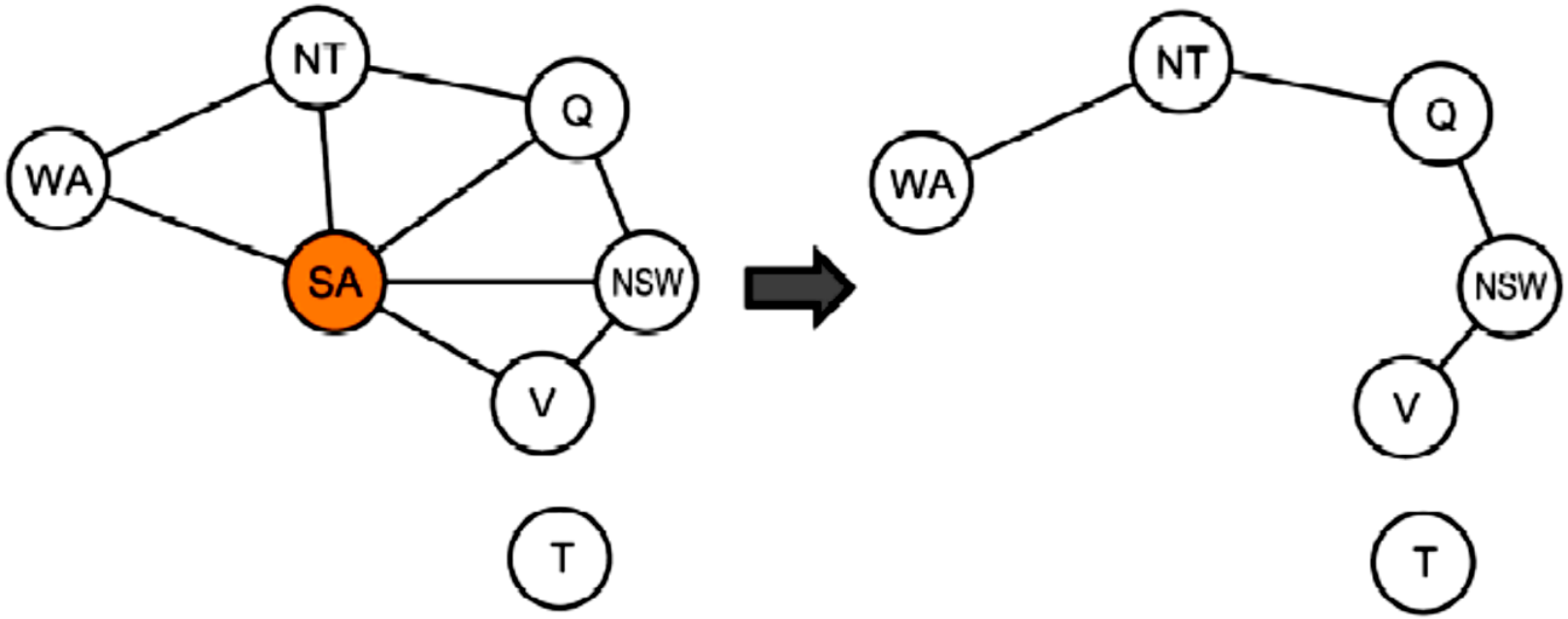
1) Assign values to specific variables (Cycle Cutset method)

1) Construct a tree-decomposition of the graph
   - Connected subproblems (subgraphs) form a tree structure

Cycle-cutset conditioning:

- Choose a subset S of variables from the graph so that graph without S is a tree
  - S = “cycle cutset”
- For each possible consistent assignment for S
  - Remove any inconsistent values from remaining variables that are inconsistent with S
  - Use tree-structured CSP to solve the remaining tree-structure
    - If it has a solution, return it along with S
    - If not, continue to try other assignments for S

Nearly Tree Structured CSPs:



- Conditioning: instantiate a variable, prune its neighbors' domains
- Cutset conditioning: instantiate (in all ways) a set of variables such that the remaining constraint graph is a tree
- Cutset size c gives runtime $O((d^c) (n-c) d^2)$, very fast for small c

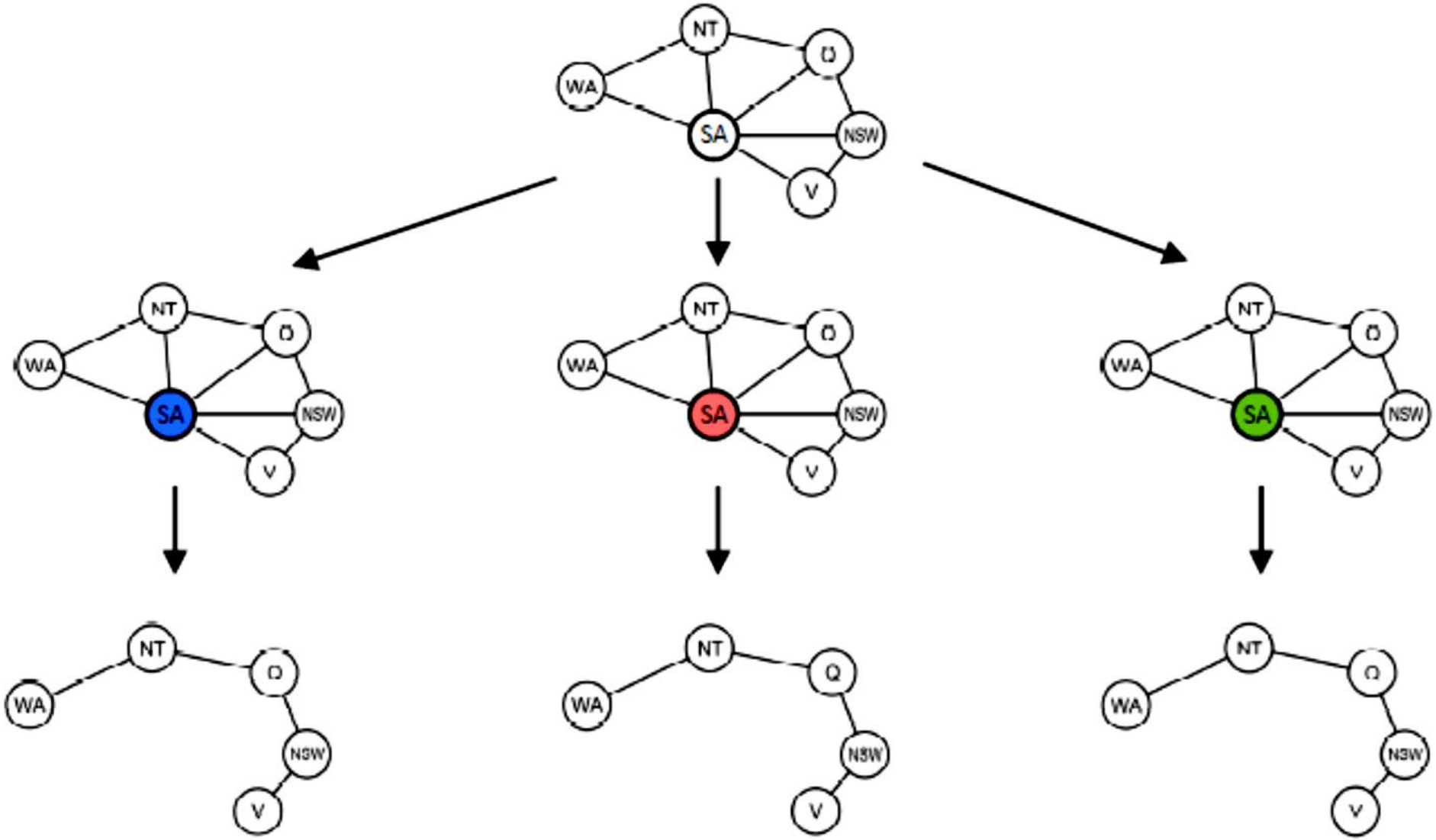
Cutset Conditioning in Action:

Choose a cutset

Instantiate the cutset (all possible ways)

Compute residual CSP for each assignment

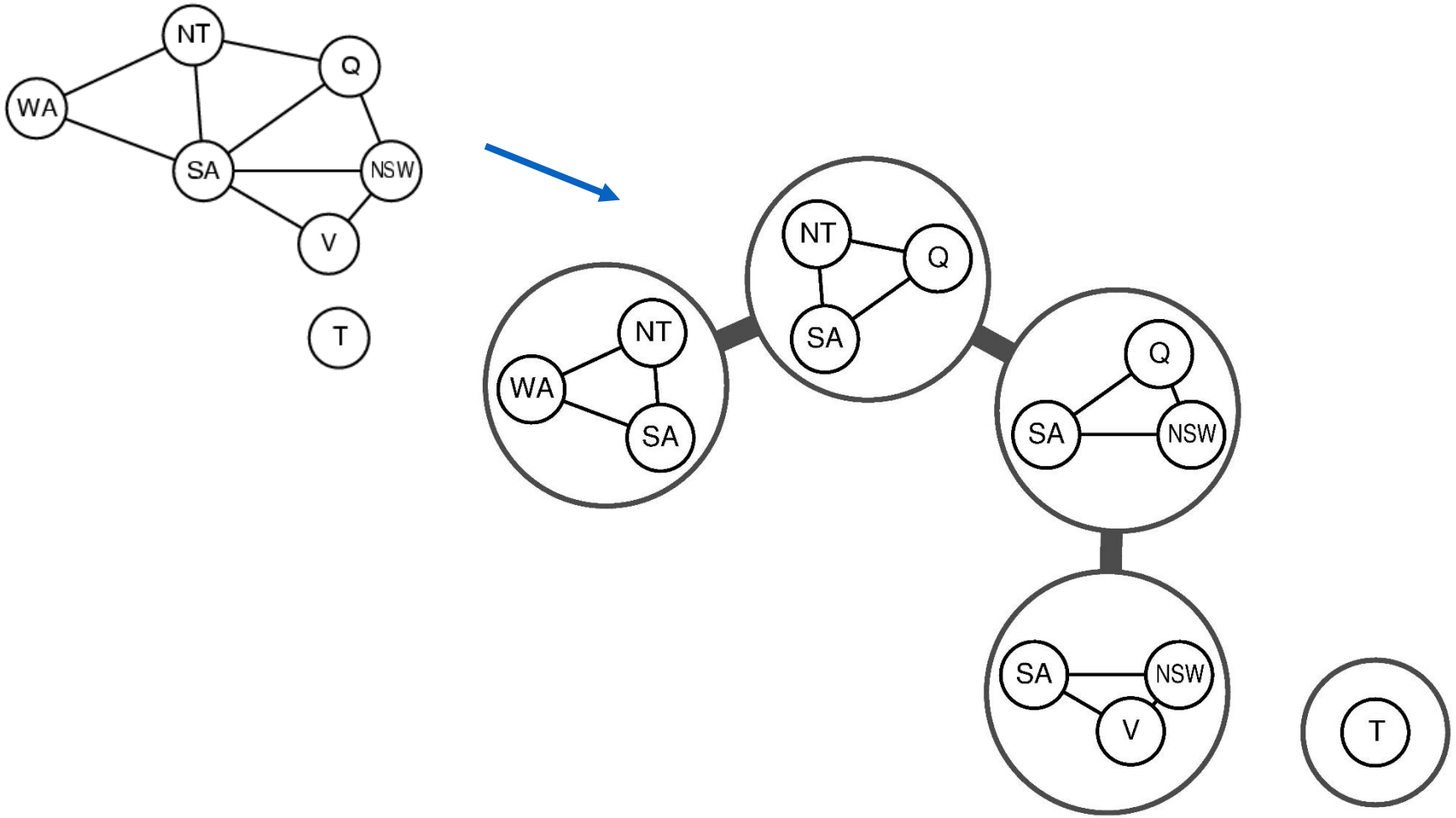
Solve the residual CSPs (tree structured)



Finding the optimal cutset

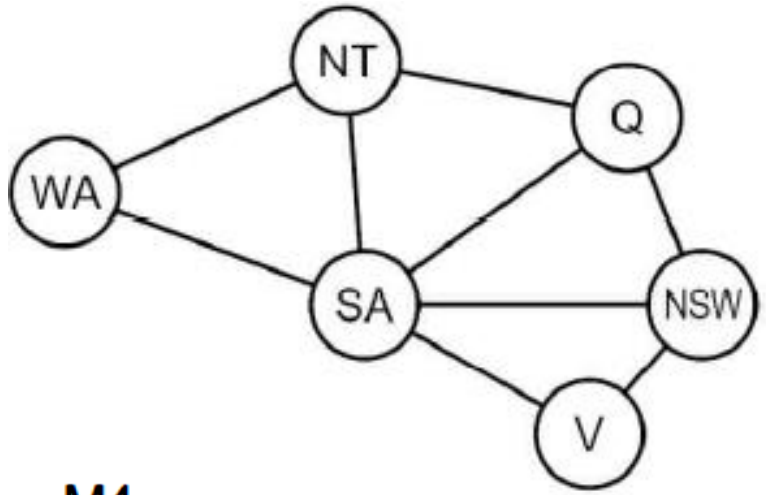
- If c is small, this technique works very well
- However, finding smallest cycle cutset is NP-hard
  - But there are good approximation algorithms

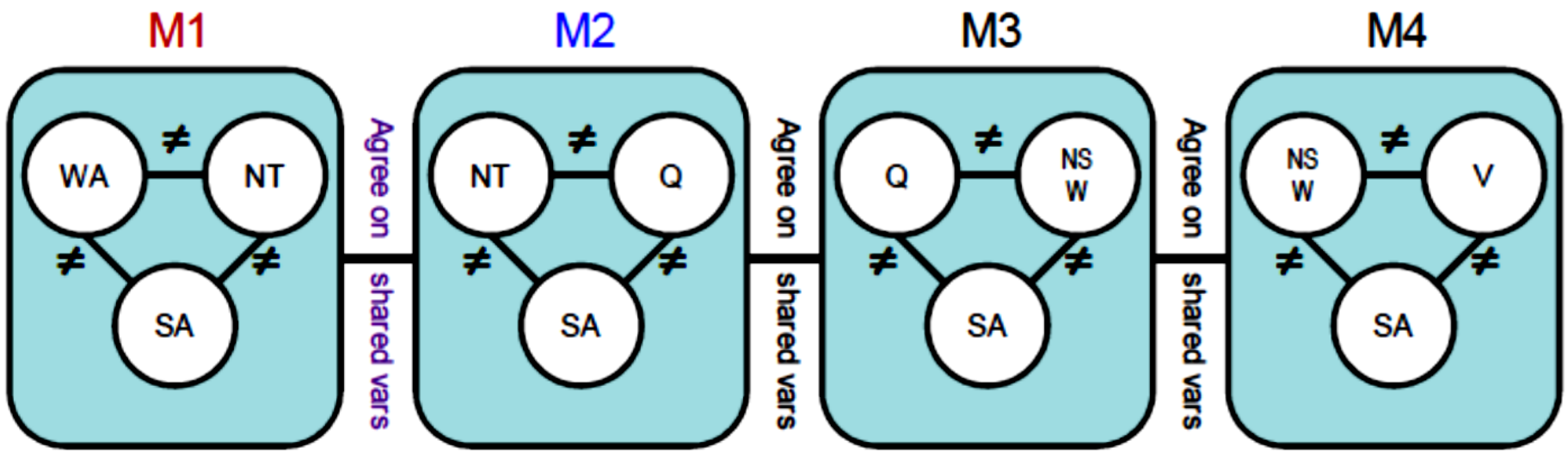
Tree Decomposition:



Note: **Just be aware but will not be included in exam**

- Idea: create a tree-structured graph of mega-variables
- Each mega-variable encodes part of the original CSP
- Subproblems overlap to ensure consistent solutions





{(WA=r,SA=g,NT=b), (WA=b,SA=r,NT=g), …}

{(NT=r,SA=g,Q=b), (NT=b,SA=g,Q=r), …}

Agree: (M1,M2) ∈ {((WA=g,SA=g,NT=g), (NT=g,SA=g,Q=g)), …}

Tree Decomposition Algorithm

- View each subproblem as a "super-variable"
  - Domain = set of solutions for the subproblem
  - Obtained by running a CSP on each subproblem

  - E.g., 6 solutions for 3 fully connected variables in map problem

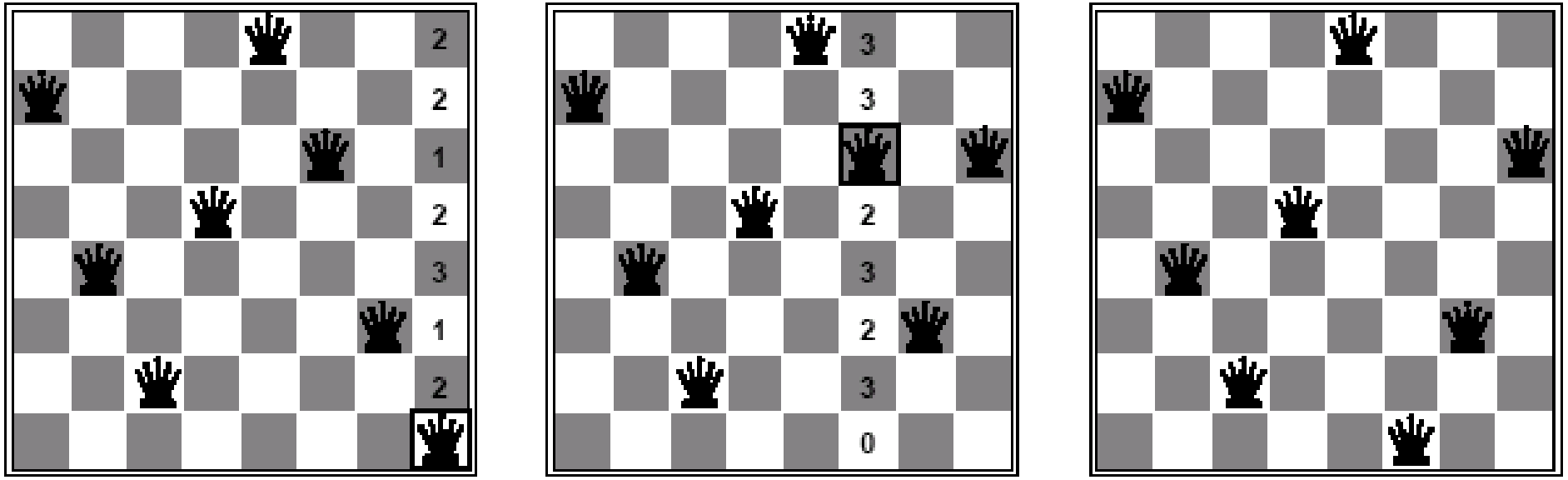
- Now use the tree CSP algorithm to solve the constraints connecting the subproblems
  - Declare a subproblem a root node, create tree
  - Backward and forward passes

- Example of "divide and conquer" strategy

Local Search a.k.a. Iterative Improvement

- Use complete-state representation
  - Initial state = all variables assigned values
  - Successor states = change 1 (or more) values
- For CSPs
  - allow states with unsatisfied constraints (unlike backtracking)
  - operators reassign variable values
- Variable selection: randomly select any conflicted variable
- Value selection: min-conflicts heuristic
  - Select new value that results in a minimum number of conflicts with the other variables

Min-conflicts Example:



- A two-step solution for an 8-queens problem using min-conflicts heuristic
- At each stage a queen is chosen for reassignment in its column
- The algorithm moves the queen to the min-conflict square breaking ties randomly.

Very powerful in practice: A million-queens in 50 steps average

Time to schedule a week of Hubble Telescope observations cut from 3 weeks to 10 minutes!

Comparison of CSP algorithms on different problems

Problem	Backtracking	BT+MRV	Forward Checking	FC+MRV	Min-Conflicts
USA	(> 1,000K)	(> 1,000K)	2K	60	64
n-Queens	(> 40,000K)	13,500K	(> 40,000K)	817K	4K
Zebra	3,859K	1K	35K	0.5K	2K
Random 1	415K	3K	26K	2K	
Random 2	942K	27K	77K	15K	

Median number of consistency checks over 5 runs to solve problem

Parentheses -> no solution found

USA: 4 coloring

n-Queens: n ranges from 2 to 50

CSPs Only Need Binary Constraints!!
Binarization of arbitrary Non-Binary Constraints

$X \in \{1, 3, 5\}$

$Y \in \{2, 4, 6\}$

$Z \in \{1, 2, 3, 4, 5\}$

$X \leq Y$

$X + Y = Z$

For each non-binary constraint, create an encapsulated variable.

Form the Cartesian Product of the domains of the associated variables,

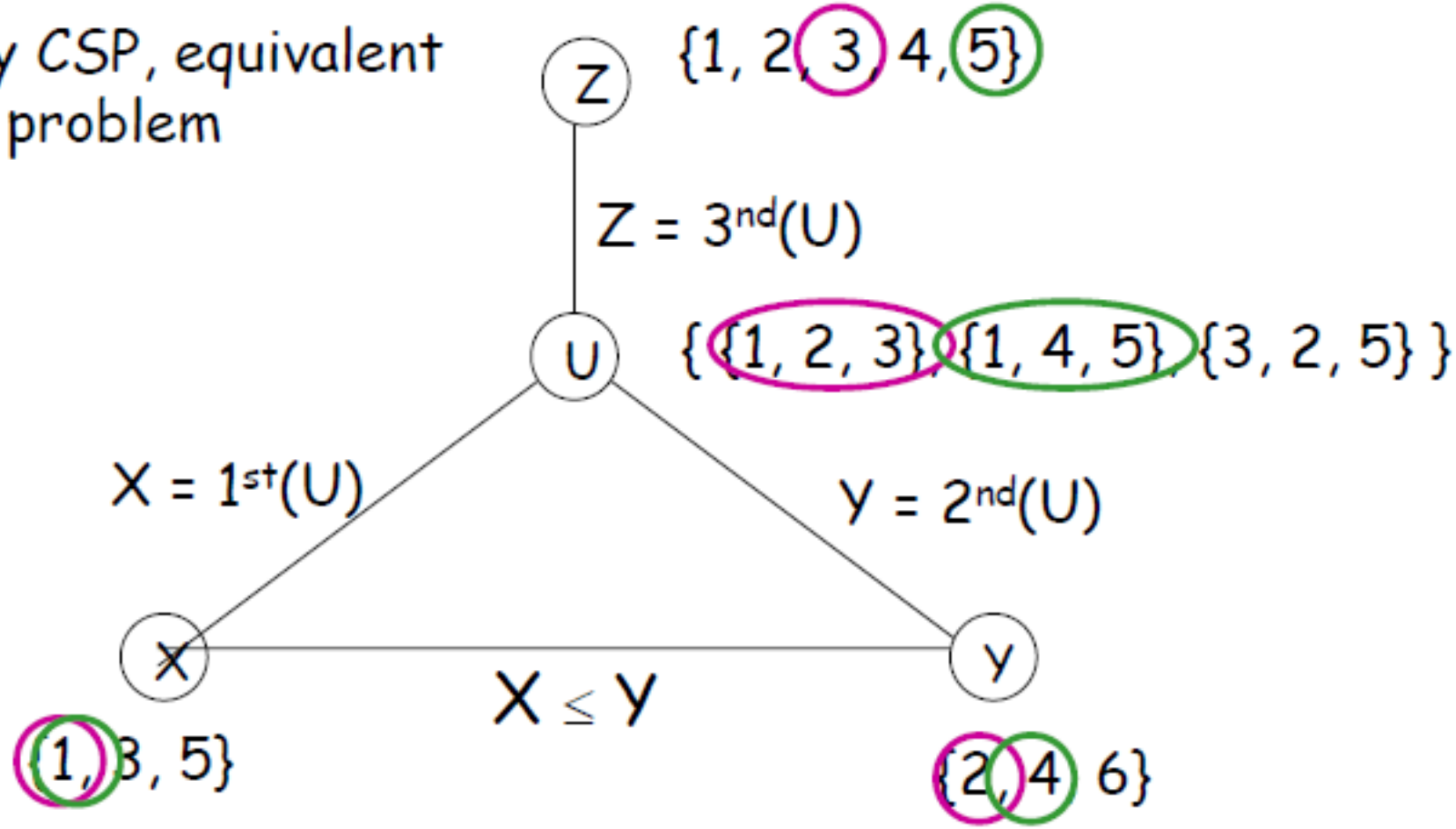
Select those elements which obey the constraint

Cartesian product of X, Y and Z is $\{1, 3, 5\} \times \{2, 4, 6\} \times \{1, 2, 3, 4, 5\}$

The domain of U is a sub-set of the Cartesian product

$U \in \{ \{1, 2, 3\}, \{1, 4, 5\}, \{3, 2, 5\} \}$

This is a binary CSP, equivalent to the original problem



Preserves the original variables

so solution to original problem can be obtained directly from the solution to this problem