Outline

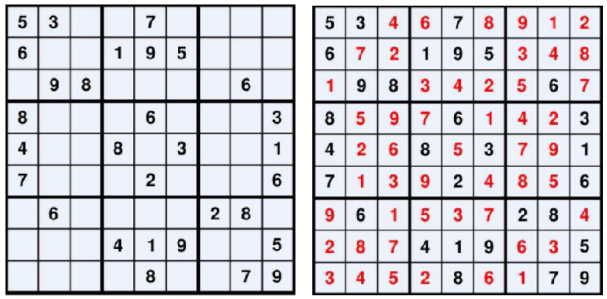
- Best-first search
  - Greedy best-first search
  - A* search
  - Heuristics
- Local search algorithms
  - Hill-climbing search
  - Beam search
  - Simulated annealing search
  - Genetic algorithms
- Constraint Satisfaction Problems
  - Constraints
  - Constraint Propagation
  - Backtracking Search
  - Local Search

Constraint Satisfaction Problems

Special Type of search problem:

- state is defined by variables X_i with values from domain D_i
- goal test is a set of constraints specifying allowable combinations of values for subsets of variables

- Examples:
  - Sudoku

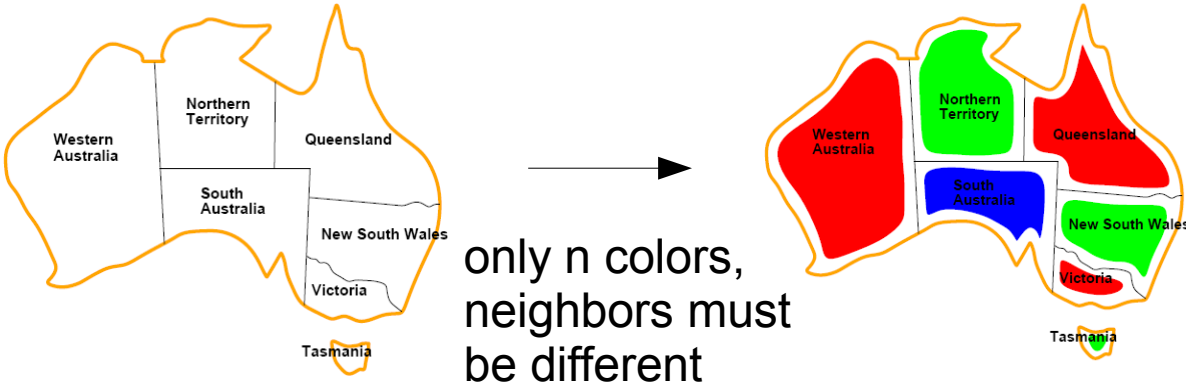


  - cryptarithmetic puzzle

```
  SEND
+ MORE
--------
 MONEY
```

  - n-queens

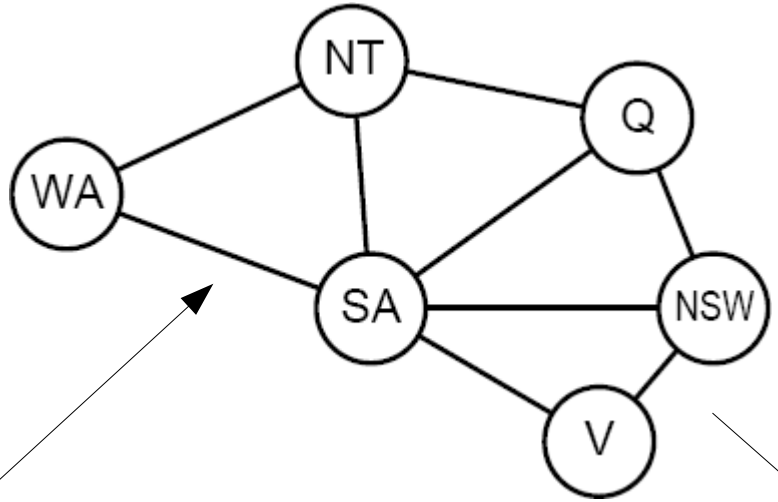
  - Graph/Map-Coloring



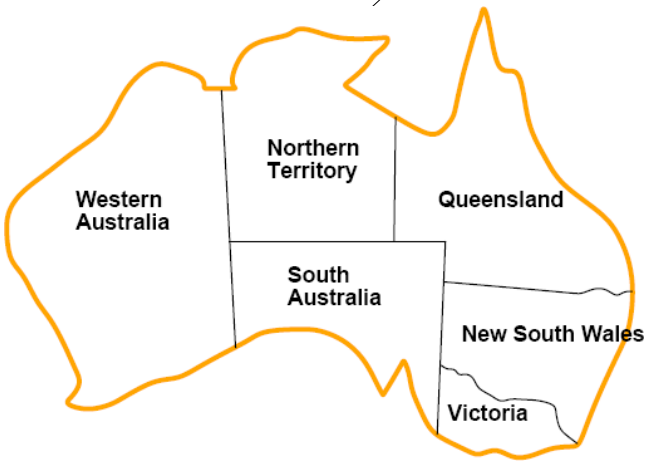
- Real-world:
  - assignment problems
  - timetables
    - classes, lecturers rooms, studies
  - ...

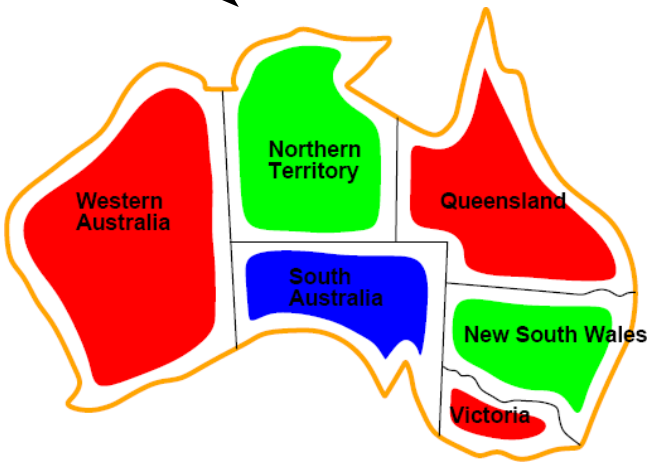
Constraint Graph

- nodes are variables
- edges indicate constraints between them



Two neighboring nodes must not have the same color

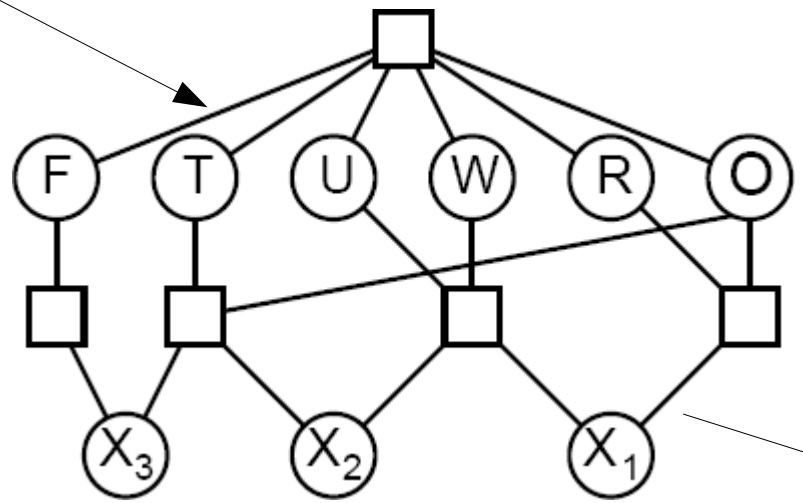




Constraint Graph

- nodes are variables
- edges indicate constraints between them

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



Connected nodes are involved in (in-)equations:

$$2 \cdot O = 10 \cdot X_1 + R$$
$$2 \cdot W + X_1 = 10 \cdot X_2 + U$$
$$2 \cdot T + X_2 = 10 \cdot X_3 + O$$
$$F = X_3$$
$$F \neq T \neq U \neq W \neq R \neq O$$

```
  7 3 4
+ 7 3 4
-------
1 4 6 8
```

Types of Constraints

- Unary constraints involve a single variable,
  - e.g., $South\ Australia \neq green$
- Binary constraints involve pairs of variables,
  - e.g., $South\ Australia \neq Western\ Australia$
- Higher-order constraints involve 3 or more variables
  - e.g., $2 \cdot W + X_1 = 10 \cdot X_2 + U$
- Preferences (soft constraints)
  - e.g., *red is better than green*
  - are not binding, but task is to respect as many as possible

→ constrained optimization problems

Solving CSP Problems

Two principal approaches:

- Constraint Propagation:
  - maintain a set of possible values D_i for each variable X_i
  - try to reduce the size of D_i by identifying values that violate some constraints
- Search:
  - successively assign values to variable
  - check all constraints
  - if a constraint is violated → backtrack
  - until all variables have assigned values

Constraint Propagation - Sudoku

- Problem
  - CSP with 81 variables
- Constraints
  - some values are assigned in the start (unary constraints)
  - 27 constraints on 9 values that must all be different

    (9 rows, 9 columns, 9 squares)
- Constraint Propagation
  - People often write a list of possible values into empty fields
  - try to successively eliminate values
- Status
  - Automated constraint solvers can solve the hardest puzzles in no time

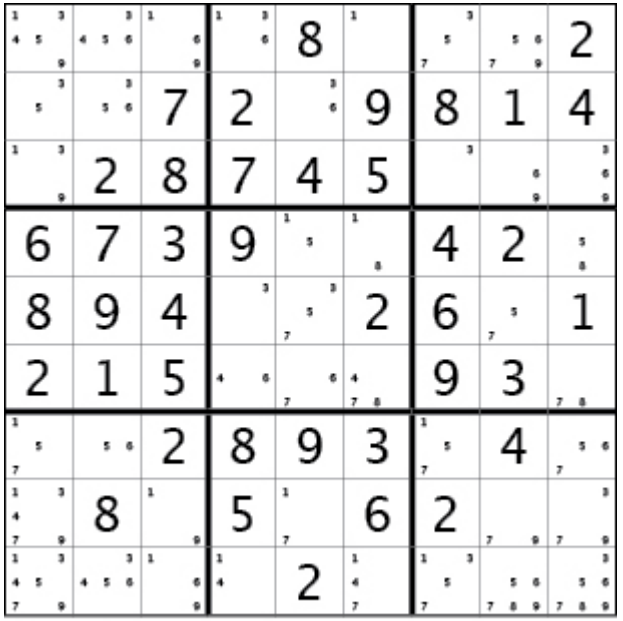


Figure 6

Node Consistency

Node Consistency

- the possible values of a variable must conform to all unary constraints
- can be trivially enforced
- Example:
  - Soduko: Some nodes are constrained to a single value

More General Idea: Local Consistency

- make each node in the graph consistent with its neighbors
- by (iteratively) enforcing the constraints corresponding to the edges

Arc Consistency

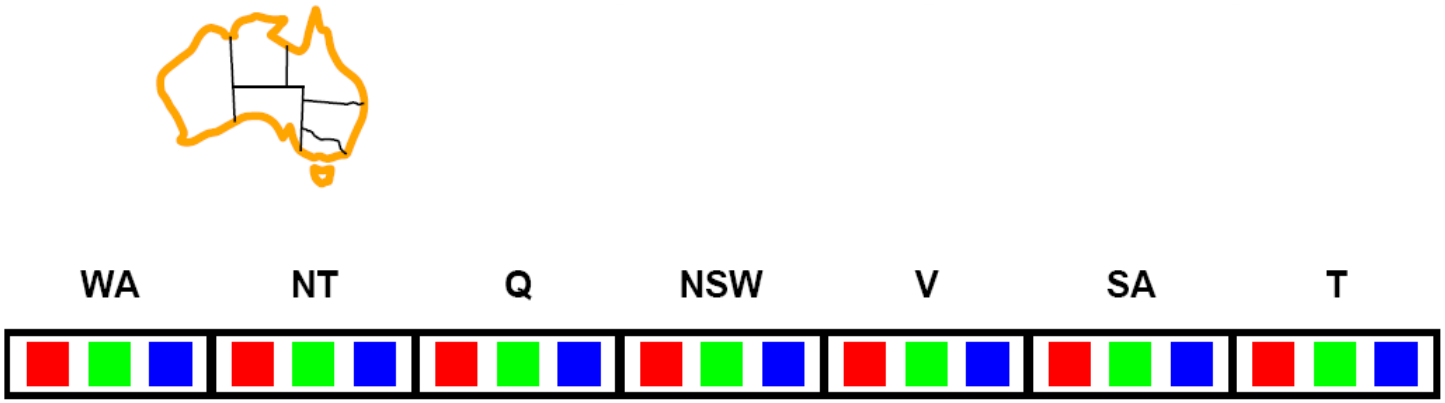
- every domain must be consistent with the neighbors:

A variable X_i is arc-consistent with a variable X_j if
- for every value in its domain D_i
- there is some value in D_j
- that satisfies the constraint on the arc (X_i, X_j)

- can be generalized to n-ary constraints
  - each tuple involving the variable X_i has to be consistent

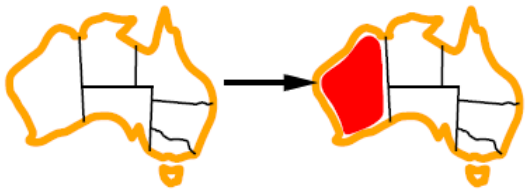
Forward Checking

- Idea: establish arc consistency for every new variable
  - keep track of remaining legal values for unassigned variables
  - terminate search when any variable has no more legal values



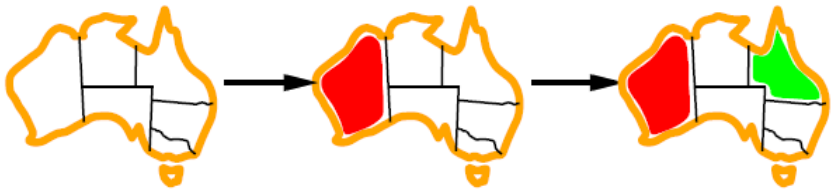
Forward Checking

- Idea:
  - keep track of remaining legal values for unassigned variables
  - terminate search when any variable no legal values



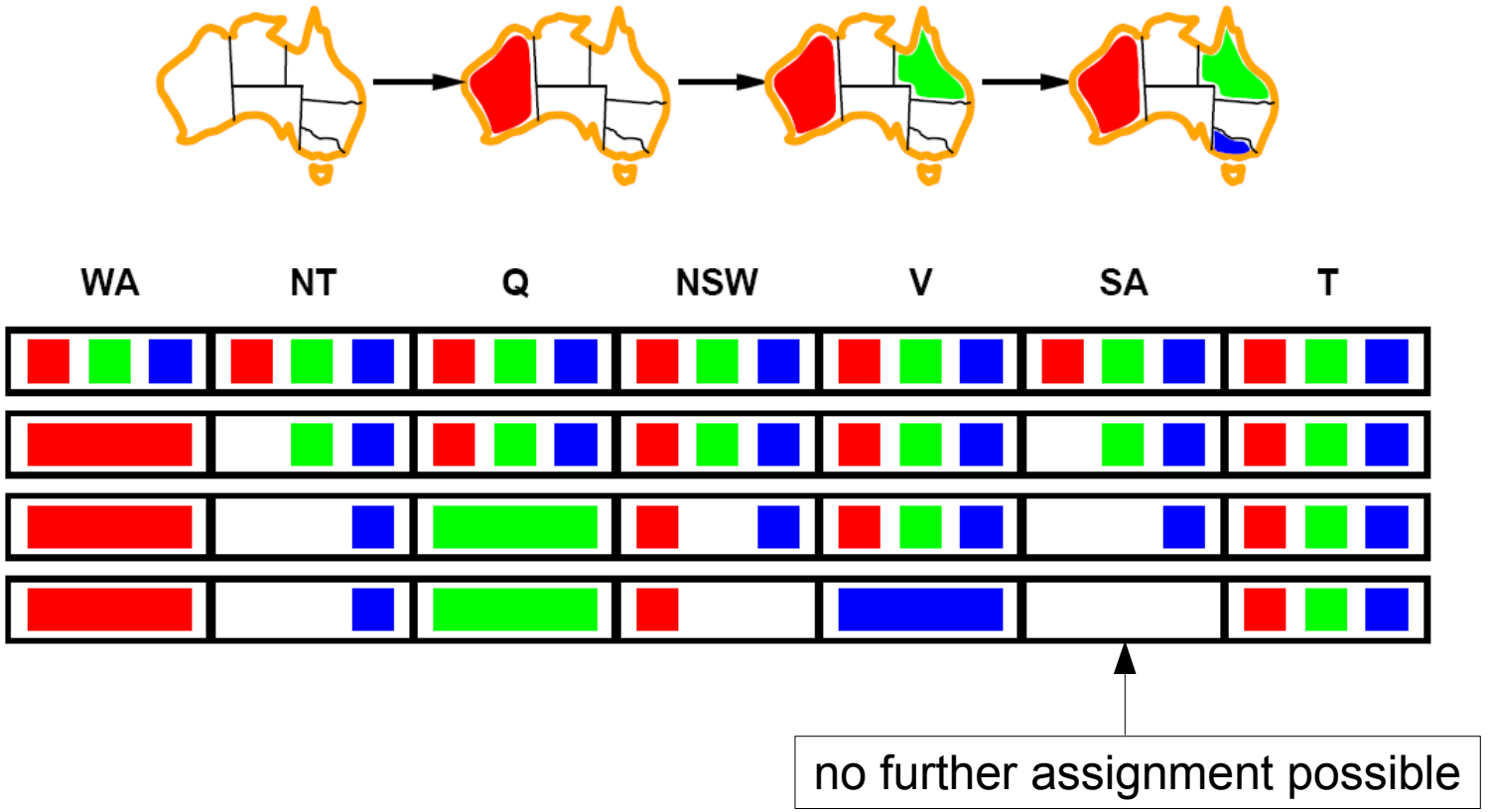
Forward Checking

- Idea:
  - keep track of remaining legal values for unassigned variables
  - terminate search when any variable no legal values



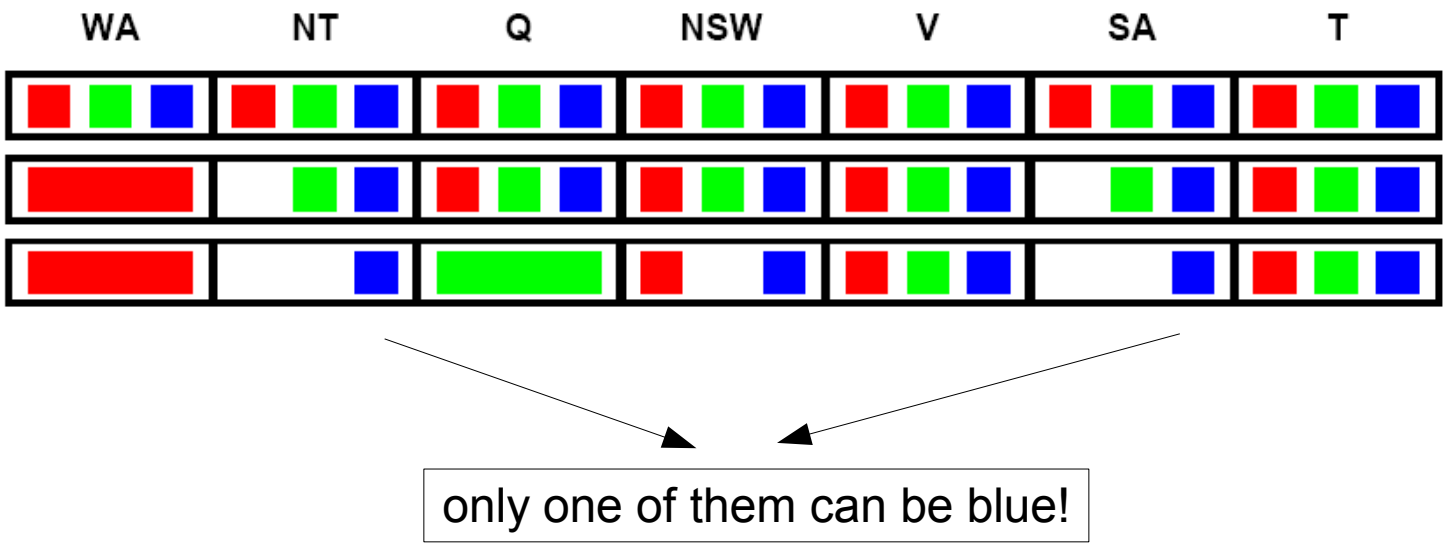
Forward Checking

- Idea:
  - keep track of remaining legal values for unassigned variables
  - terminate search when any variable no legal values



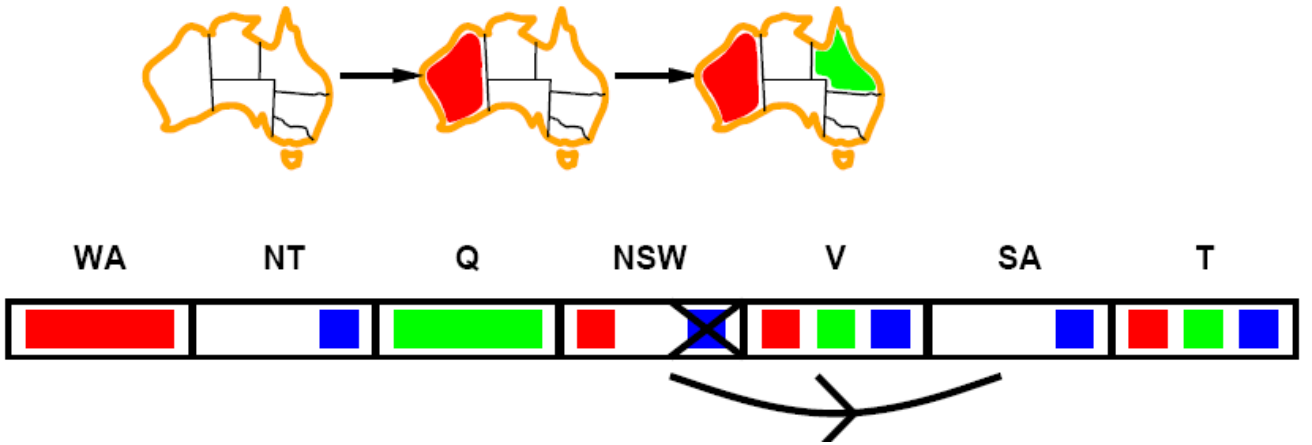
Constraint Propagation

- Problem:
  - forward checking propagates information from assigned to unassigned variables
  - but doesn't look ahead to provide early detection for all failures

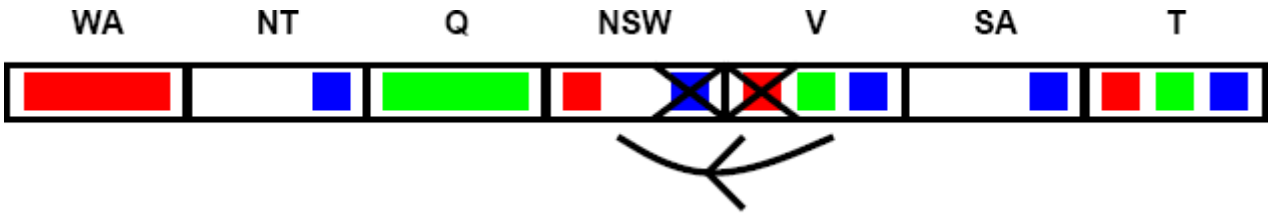


Maintaining Arc Consistency (MAC)

- After each new assignment of a value to a variable, possible values of the neighbors have to be updated:



- If one variable (NSW) looses a value (blue), we need to recheck its neighbors as well:



Arc Consistency Algorithm

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

  while *queue* is not empty **do**
    $(X_i, X_j) \leftarrow$ REMOVE-FIRST(*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*

If X loses a value, neigbors of X need to be rechecked.

---

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*

- Run-time: $O(n^2d^3)$ (can be reduced to $O(n^2d^2)$)
  more efficient than forward checking (see later)

Path Consistency

- Arc Consistency is often sufficient to
  - solve the problem (all domains have size 1)
  - show that the problem cannot be solved (some domains empty)
- but may not be enough
  - there is always a consistent value in the neighboring region

→ Path consistency

- tightens the binary constraints by considering triples of values

A pair of variables (X_i, X_j) is path-consistent with X_m if
- for every assignment that satisfies the constraint on the arc (X_i, X_j)
- there is an assignment that satisfies the constraints on the arcs (X_i, X_m) and (X_j, X_m)

- Algorithm AC-3 can be adapted to this case (known as PC-2)

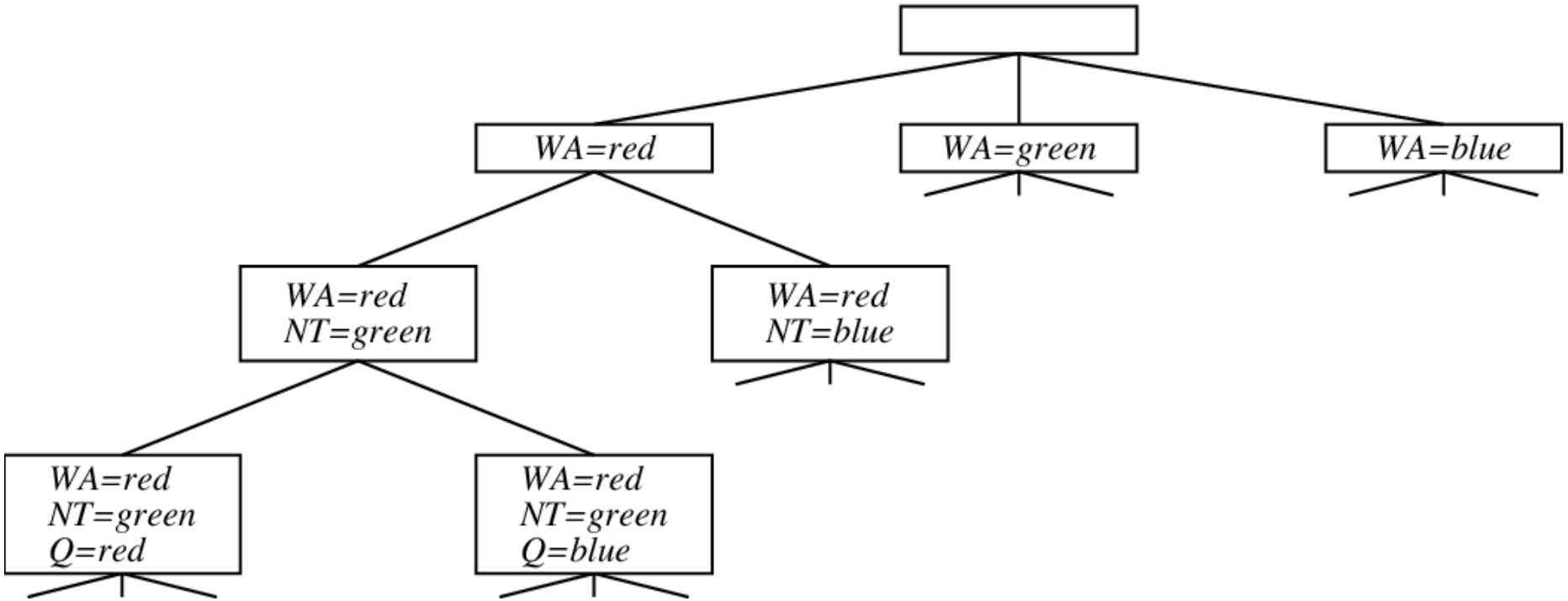
k-Consistency

- The concept can be generalized so that a set of k values need to be consistent
  - 1-consistency = node consistency
  - 2-consistency = arc consistency
  - 3-consistency = path consistency
  - ....
- May lead to faster solution ($O(n^2 d)$)
  - but checking for k-Consistency is exponentional in k in the worst case
- therefore arc consistency is most frequently used in practice

Sudoku

- simple puzzles can be solved with AC-3
  - the 9-valued AllDiff constraints can be converted into pairwise binary constraints (36 each)
  - therefore 27x36 = 972 arc constraints
- somewhat more with PC-2
  - there are 255,960 path constraints
- however, not all problems can be solved with constraint progapagation alone
  - to solve all puzzles we need a bit of search

Search Tree for CSP



Backtracking Search

- CSP are typically solved with backtracking
  - add one constraint at a time without conflict
  - succeed if a legal assignment is found

```
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
```

Worst-Case Complexity of Backtracking Search

- Assumptions
  - we have n variables
    → all solutions are a depth n in the search tree
  - all variables have v possible values
- Then
  - at level 1 we have $n \cdot v$ possible assignments
    (we can choose one of n variables and one of v values for it)
  - at level 2, we have $(n-1) \cdot v$ possible assignments for each previously assigned variable
    (we can choose one of the remaining $n-1$ variables and one of the v values for it)
  - In general: branching factor at depth l: $(n-l+1) \cdot v$
- Hence
  - The search tree has $n! v^n$ leaves

Note: If the order of variable assigments does not matter the $n!$ may be saved

→ heuristics for selecting variables (SELECTUNASSIGNEDVARIABLE) and for ordering values (ORDERDOMAINVALUES)

General Heuristics for CSP

- Domain-Specific Heuristics
  - Depend on the particular characteristics of the problem
  - Obviously, a heuristic for the 8-puzzle can not be used for the 8-queens problem
- General-purpose heuristics
  - For CSP, good general-purpuse heuristics are known:
  - Mininum Remaining Values Heuristic
    - choose the variable with the fewest consistent values
  - Degree Heuristic
    - choose the variable that imposes the most constraints on the remaining values

SelectUnassignedVariable

  - Least Constraining Value Heuristic
    - Given a variable, choose the value that rules out the fewest values in the remaining variables

OrderDomainValues

  - used in this order, these three can greatly speed up search
    - e.g., n-queens from 25 queens to 1000 queens

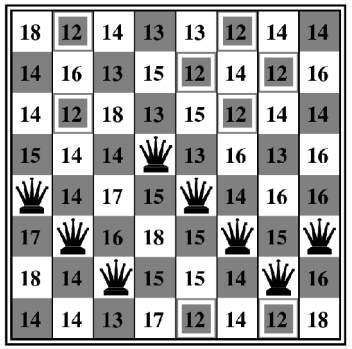
Integrating Constraint Propagation and Backtracking Search

- Performance of Backtracking can be further sped up by integrating constraint propagation into the search

- Key idea:
  - each time a variable is assigned, a constraint propagation algorithm is run in order to reduce the number of choice points in the search

- Possible algorithms
  - Forward Checking
  - AC-3, but initial queue of constraints only contains constraints with the variable that has been changed

Local Search for CSP



- Modifications for CSPs:
  - work with complete states
  - allow states with unsatisfied constraints
  - operators reassign variable values
- Min-conflicts Heuristic:
  - randomly select a conflicted variable
  - choose the value that violates the fewest constraints
  - hill-climbing with $h(n) =$ # of violated constraints

Min-conflicts is the heuristic that we studied for the 8-queens problems.

- Performance:
  - can solve randomly generated CSPs with a high probability
  - except in a narrow range of

$$R = \frac{\text{number of constraints}}{\text{number of variables}}$$