AI基础

Lecture 6: Constraint Satisfaction Problems

Bin Yang
School of Data Science and Engineering
byang@dase.ecnu.edu.cn

[Some slides adapted from Dan Klein and Pieter Abbeel, UCB]

Lecture 5 ILOs

- Adversarial Search and Games
  - Game theory, problem settings
    - # players, deterministic vs. stochastic, zero-sum vs. general games
  - Optimal Decisions in games
    - binary outcome (win or lose): AND-OR tree search
    - Multiple outcomes: minimax search
    - Stochastic games: minimax search with chance nodes computing expected utilities
  - Alpha-Beta Pruning
  - Heuristic Alpha-Beta Tree Search
    - Cutoff, use heuristic evaluation
  - Monte Carlo Tree Search
    - Selection, Expansion, Simulation, and backpropagation
    - Exploration and exploitation

Lecture 6 ILOs

- Constraint Satisfaction Problems
    - Definition of CSPs
        - Variables, Domains, Constraints
    - Types of constrains
        - Unary, binary, high-order, preferences
    - Solving CSPs
        - Standard search
        - Backtracking
        - Problem structures
        - Local search

Outline

- Definition of CSPs
    - Variables, Domains, Constraints
    - Example CSPs
- Types of constrains
    - Unary, binary, high-order, preferences
- Solving CSPs
    - Standard search
    - Backtracking
    - Problem structures
    - Local search

Problem settings

- Standard search problems
  - State space graph
  - State is a “black box”, atomic
  - Solution: a path from the initial state to a goal state
- Constraint satisfaction problems (CSPs)
  - Factored representation
  - Each state: a set of variables, each has a value
  - Goal test: each variable has a value such that all constraints on the variables are satisfied
  - Solution: a goal state, legal assignments to all variables
- More powerful than standard search problems

Defining Constraint Satisfaction Problems

- A constraint satisfaction problem consists of three components:
  - X is a set of variables $\{X_1, X_2, \cdots, X_n\}$
  - D is a set of domains $\{D_1, D_2, \cdots, D_n\}$, one for each variable
  - C is a set of constraints that specify allowable combination of values
    - Each constraint C_j is a pair <scope, rel>
- Example
  - Variables: X_1 and X_2
  - Domains: both variables have the domain {1, 2, 3}
  - A constraint saying that X_1 must be greater than X_2
    - $<(X_1, X_2), \{(3, 1), (3, 2), (2, 1)\}>$ or $<(X_1, X_2), X_1>X_2 >$
- Simple example of a formal representation language
  - Allows useful general-purpose algorithms with more power than standard search algorithms

Defining Constraint Satisfaction Problems

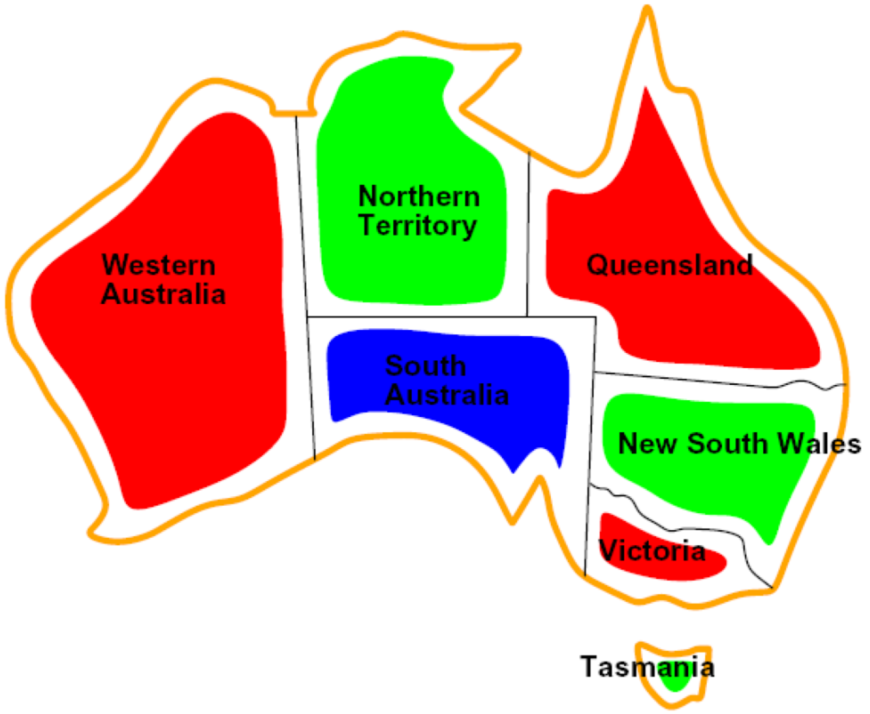
- A constraint satisfaction problem consists of three components X, D, and C.
- CSPs deal with assignments of values to variables
  - $\{X_i=v_i, X_j=v_j, \cdots\}$
- Some concepts on assignments
  - Consistent/legal assignment: an assignment that does not violate any constraints
  - Complete assignment: every variable is assigned a value
  - A solution to a CSP is a consistent, complete assignment.
    - In contrast to a solution in standard search: a path from the initial state to a goal state
  - A partial assignment is one that leaves some variables unassigned
  - A partial solution is a partial assignment that is consistent.

Example problem: Map Coloring

- Coloring each region with either red, green, or blue in such a way that no two neighboring regions have the same color.
- X={WA, NT, SA, Q, NSW, V, T}
  - 7 variables in total.
- D={red, green, blue}
  - Each variable has the same domain.
- Constraints: adjacent regions must have different colors

$$\begin{aligned} C = \{& SA \neq WA, SA \neq NT, SA \neq Q, SA \neq NSW, SA \neq V, \\ & WA \neq NT, NT \neq Q, Q \neq NSW, NSW \neq V\}. \end{aligned}$$

- Solution
  - {WA=Red, NT=Green, Q=Red, SA=Blue, NSW=G, V=R, T=Green}



Example problem: Job-shop scheduling

- A factory schedules many tasks.
- Model each task as a variable, where the value of each variable is the time that the task starts, expressed as an integer number of minutes.
- Constraints can assert that one task must occur before another.
- We consider a small part of the car assembly, consisting of 15 tasks (representing the tasks with 15 variables):
  - install axles (front and back),
  - affix all four wheels (right and left, front and back),
  - tighten nuts for each wheel, affix hubcaps, and inspect the final assembly.

$$\begin{aligned} X \quad = \quad & \{Axle_F, Axle_B, Wheel_{RF}, Wheel_{LF}, Wheel_{RB}, Wheel_{LB}, Nuts_{RF}, \\ & Nuts_{LF}, Nuts_{RB}, Nuts_{LB}, Cap_{RF}, Cap_{LF}, Cap_{RB}, Cap_{LB}, Inspect\}. \end{aligned}$$

Example problem: Job-shop scheduling

- The axles have to be in place before the wheels are put on, and it takes 10 minutes to install an axle

$$Axle_F + 10 \le Wheel_{RF}; \quad Axle_F + 10 \le Wheel_{LF};$$
$$Axle_B + 10 \le Wheel_{RB}; \quad Axle_B + 10 \le Wheel_{LB}.$$

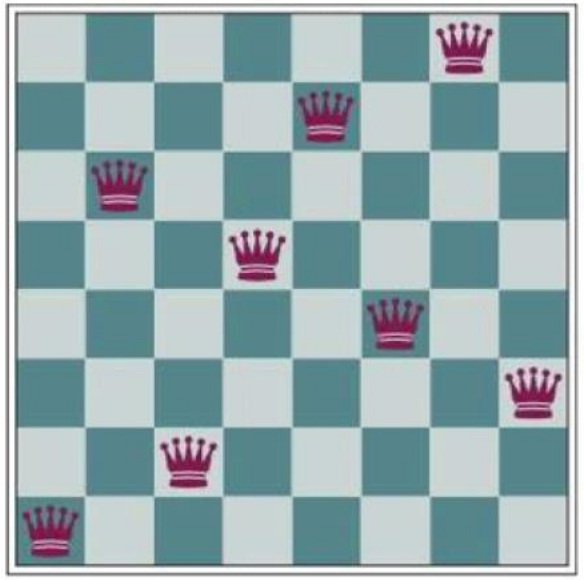
- Affix the wheel (which takes 1 minute), then tighten the nuts (2 minutes), and finally attach the hubcap

$$Wheel_{RF} + 1 \le Nuts_{RF}; \quad Nuts_{RF} + 2 \le Cap_{RF};$$
$$Wheel_{LF} + 1 \le Nuts_{LF}; \quad Nuts_{LF} + 2 \le Cap_{LF};$$
$$Wheel_{RB} + 1 \le Nuts_{RB}; \quad Nuts_{RB} + 2 \le Cap_{RB};$$
$$Wheel_{LB} + 1 \le Nuts_{LB}; \quad Nuts_{LB} + 2 \le Cap_{LB}.$$

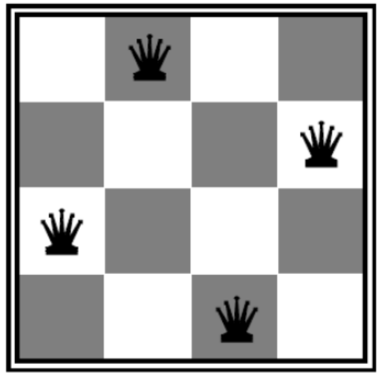
Miniquiz

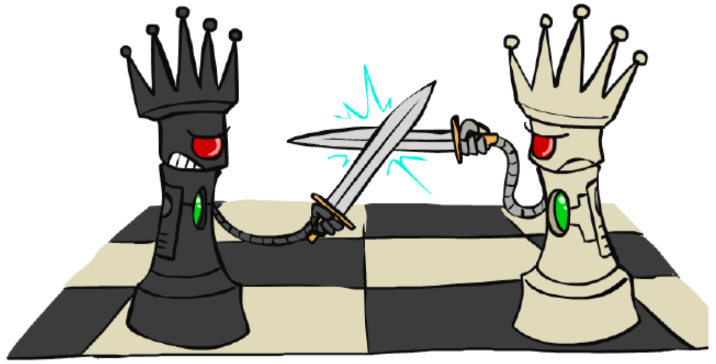
- The 8-queens problem
  - Place 8 queens on a chess board so that no queen attacks another. (A queen attacks any piece in the same row, column, or diagonal.)

- How to specify the constraints in CSP?



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





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

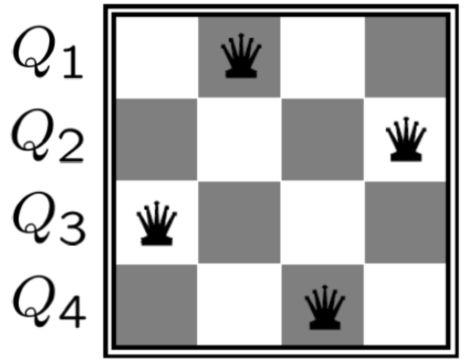
$$\sum_{i,j} X_{ij} = N$$

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

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

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

$\cdots$

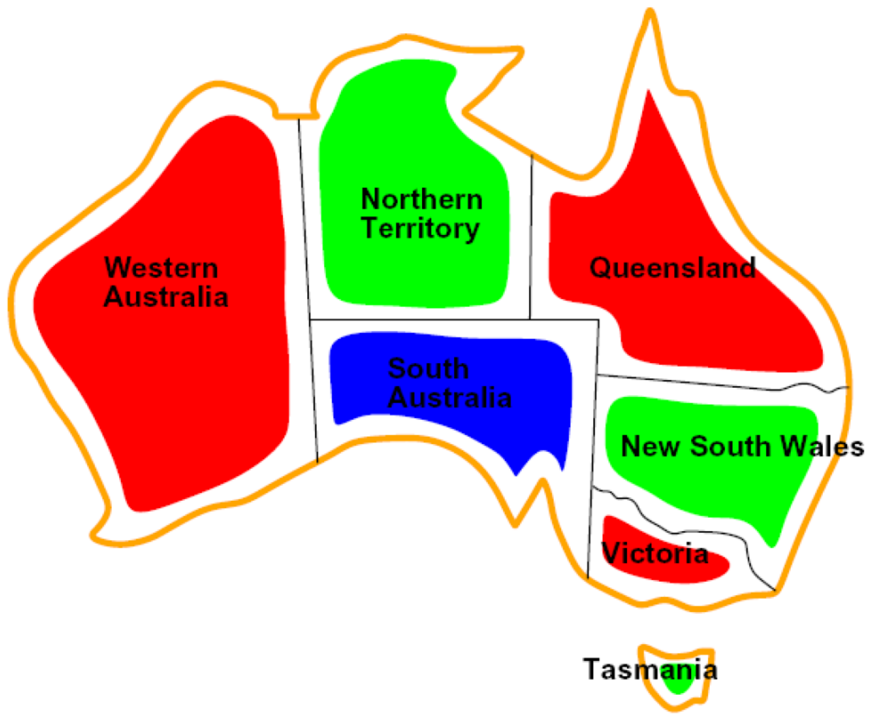


Outline

- Definition of CSPs
    - Variables, Domains, Constraints
    - Example CSPs
- Types of constrains
    - Unary, binary, high-order, preferences
- Solving CSPs
    - Standard search
    - Backtracking
    - Problem structures
    - Local search

Varieties of constraints

- Unary constraint
  - SA≠Green
- Binary constraint
  - SA≠WA

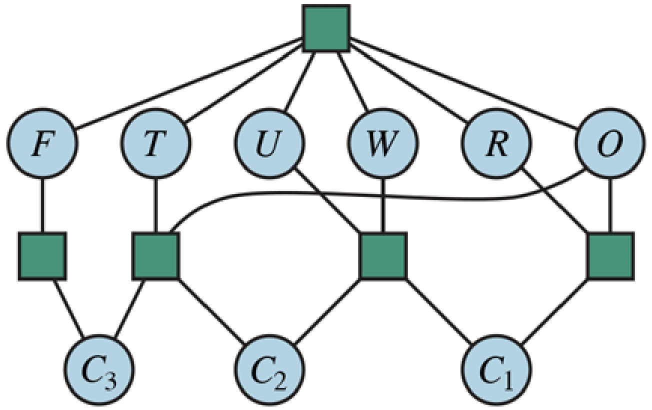


Varieties of constraints

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

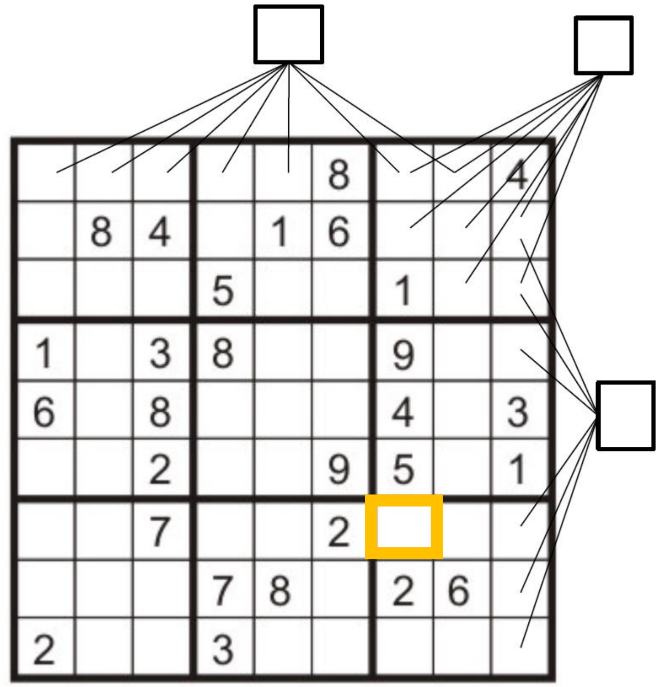
- Higher-order constraints
  - Cryptarithmetic: each letter represents a digit.
  - Domain D={0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
  - Global constraint: Alldiff(F, T, U, W, R, O)
  - n-ary constraints

$$\begin{aligned} &O + O = R + 10 \cdot C_1 \\ &C_1 + W + W = U + 10 \cdot C_2 \\ &C_2 + T + T = O + 10 \cdot C_3 \\ &C_3 = F\ , \end{aligned}$$



Varieties of constraints

- Sudoku
- Variables: Each (open) square
  - At most 81
- Domains: {1,2, 3, 4, 5, 6, 7, 8, 9}
- Constraints:
  - 9-way Alldiff for each row
  - 9-way Alldiff for each column
  - 9-way Alldiff for each region
  - or can have a bunch of pairwise inequality constraints



$Alldiff(A1,A2,A3,A4,A5,A6,A7,A8,A9)$

$Alldiff(B1,B2,B3,B4,B5,B6,B7,B8,B9)$

$\cdots$

$Alldiff(A1,B1,C1,D1,E1,F1,G1,H1,I1)$

$Alldiff(A2,B2,C2,D2,E2,F2,G2,H2,I2)$

$\cdots$

$Alldiff(A1,A2,A3,B1,B2,B3,C1,C2,C3)$

$Alldiff(A4,A5,A6,B4,B5,B6,C4,C5,C6)$

$\cdots$

Varieties of constraints

- Preference constrains
  - University class-scheduling
  - Absolute constraints: no professor can teach two classes at the same time.
  - Preference constraints: Prof. R might prefer teaching in the morning, whereas Prof. N prefers teaching in the afternoon.
  - A schedule that has Prof. R teaching at 2 p.m. would still be an allowable solution but would not be an optimal one.
  - Often representable by a cost for each variable assignment.
    - Prof R teaching at 2 p.m. costs 2
    - Prof R teaching at 8 a.m. costs 1
  - Constrained optimization problems (COPs)

Real world CSPs

- Scheduling problems: e.g., when can we all meet?
- Timetable problems: e.g., which class is offered when and where?
- Assignment problems: e.g., who teaches what class
- Hardware configuration
- Transportation scheduling
- Factory scheduling
- Circuit layout
- ⋯ lots more!

- Many real-world problems involve real-valued variables

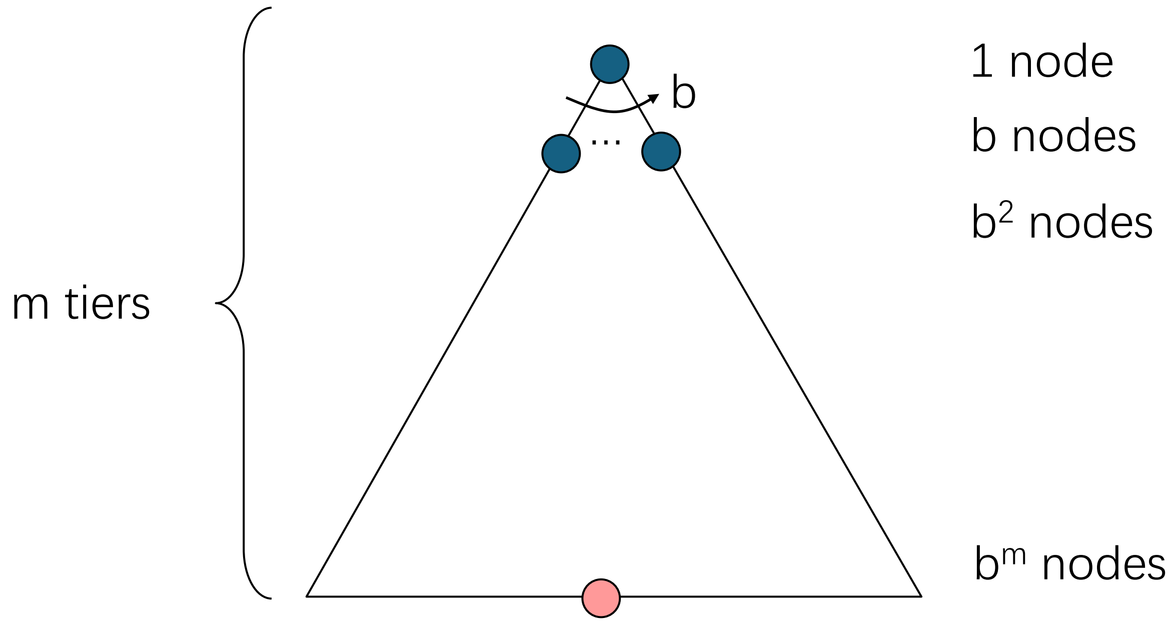
Outline

- Definition of CSPs
  - Variables, Domains, Constraints
- Types of constrains
  - Unary, binary, high-order, preferences
- Solving CSPs
  - Standard search
  - Backtracking
  - Problem structures
  - Local search

Solving CSPs: standard search

- Search Tree
  - Branching factor b
  - Maximum depth m

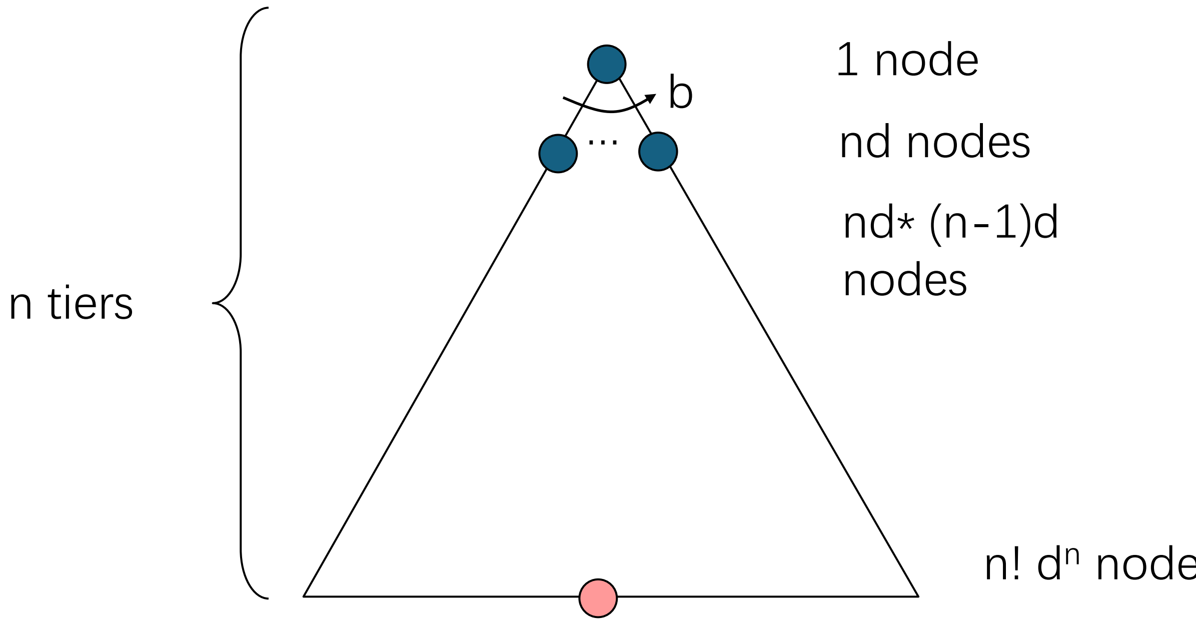
- For a CSP has n variables of domain size d
- Mini-quiz: what is the search tree? How many nodes in the bottom level, i.e., # of leaves?



Solving CSPs: standard search

- Search Tree
    - Branching factor b
    - Maximum depth m

- For a CSP has n variables of domain size d
- All complete assignments are leaf nodes at depth n
    - Maximum depth is n
- Branching factors
    - Level 1: nd
    - Level 2: (n-1)d
    - Level 3: (n-2)d
    - …
    - Level n-1: 2d
    - Level n: d
- Leaf nodes: n! d^n
- Complete assignments d^n
- Why this happens?
    - Commutativity.



Outline

- Definition of CSPs
  - Variables, Domains, Constraints
- Types of constrains
  - Unary, binary, high-order, preferences
- Solving CSPs
  - Standard search
  - Backtracking
  - Problem structures
  - Local search

Solving CSPs: backtracking search

- Backtracking search is the basic uninformed algorithm for solving CSPs
- One variable at a time
    - Variable assignments are commutative, so fix ordering
    - [WA = red then NT = green] same as [NT = green then WA = red]
    - Only need to consider assignments to a single variable at each step
- Check constraints as you go
    - Consider only values which do not conflict with 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

Backtracking

- Backtracking=
  - DFS
  - Variable ordering
  - Fail-on-violation

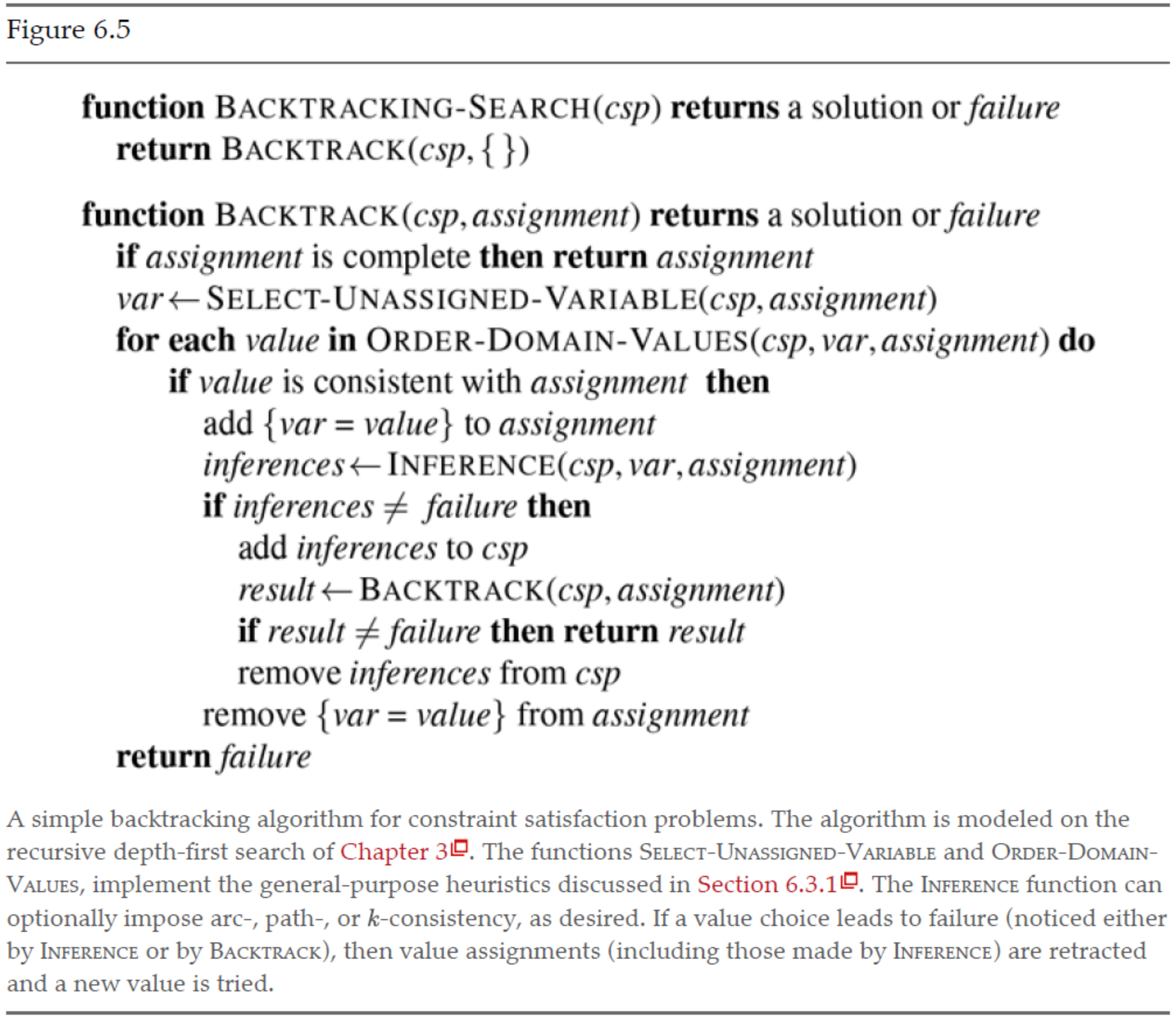
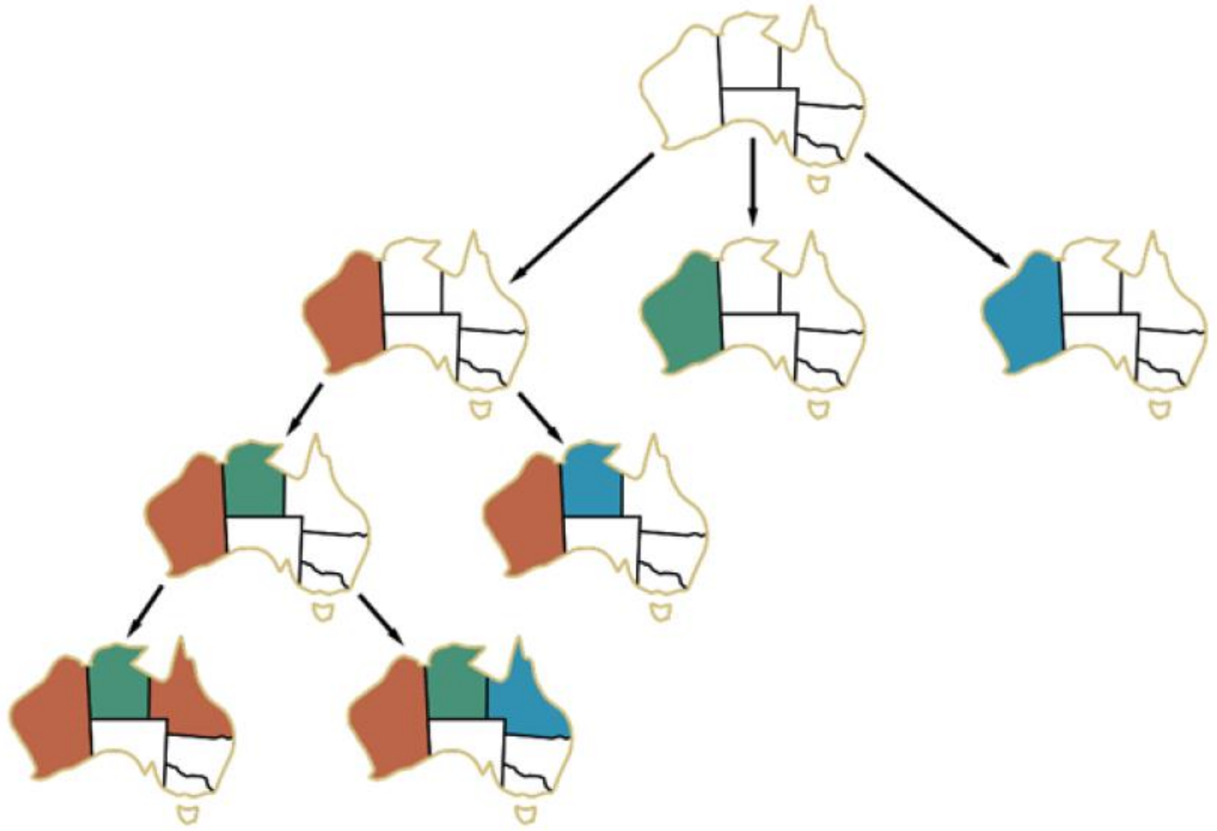


Figure 6.5

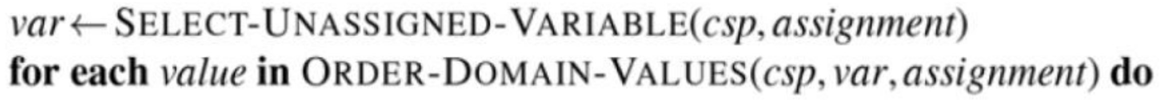
```
function BACKTRACKING-SEARCH(csp) returns a solution or failure
   return BACKTRACK(csp, { })

function BACKTRACK(csp, assignment) returns a solution or failure
   if assignment is complete then return assignment
   var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)
   for each value in ORDER-DOMAIN-VALUES(csp, var, assignment) do
      if value is consistent with assignment then
         add {var = value} to assignment
         inferences ← INFERENCE(csp, var, assignment)
         if inferences ≠ failure then
            add inferences to csp
            result ← BACKTRACK(csp, assignment)
            if result ≠ failure then return result
            remove inferences from csp
         remove {var = value} from assignment
   return failure
```

A simple backtracking algorithm for constraint satisfaction problems. The algorithm is modeled on the recursive depth-first search of Chapter 3. The functions SELECT-UNASSIGNED-VARIABLE and ORDER-DOMAIN-VALUES, implement the general-purpose heuristics discussed in Section 6.3.1. The INFERENCE function can optionally impose arc-, path-, or k-consistency, as desired. If a value choice leads to failure (noticed either by INFERENCE or by BACKTRACK), then value assignments (including those made by INFERENCE) are retracted and a new value is tried.



Improving Backtracking

- In standard search: Uninformed search can be improved by domain-specific heuristics
- Backtracking can be improved with domain-independent heuristics that take advantage of the factored representations of CSPs
- Ordering:
  - Which variable should be assigned next?
  - In which order should its values be tried?

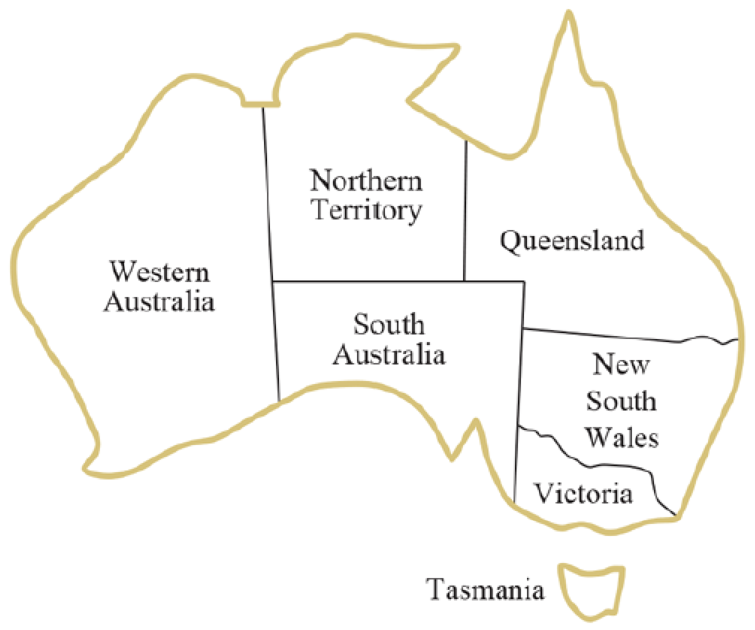


```
var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)
for each value in ORDER-DOMAIN-VALUES(csp, var, assignment) do
```

- Filtering:
  - Can we detect inevitable failure early?
- Structure:
  - Can we exploit the problem structure?

Ordering: variables

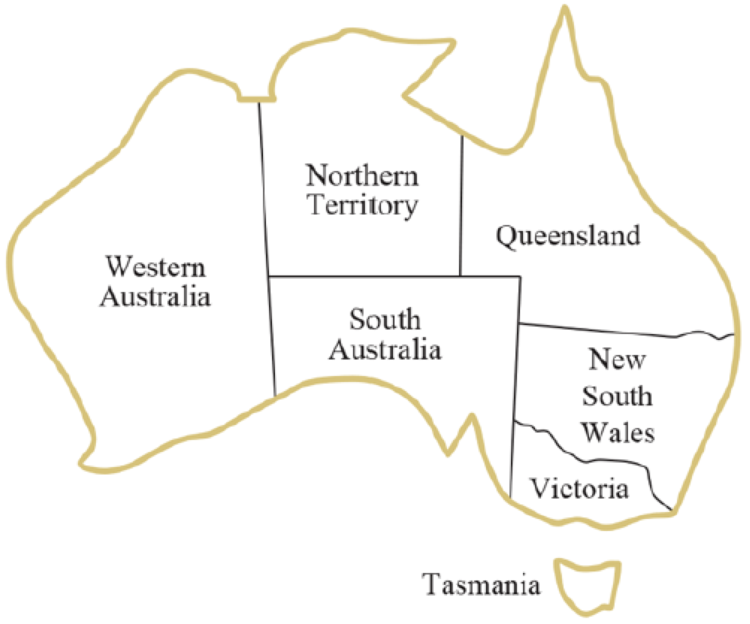
- Static ordering
  - X_1, X_2, …
- Random ordering
- When we assign WA=Red and NT=Green
  - Which variable to choose next?
  - Only one choice for SA=Blue
  - Then, Q, NSW, V are all enforced.
- Variable Ordering: Minimum remaining values (MRV):
  - Choose the variable with the fewest legal values left in its domain
  - SA=1, Q=2, NSW=3, V=3, T=3
- Why min rather than max?
  - most constrained variable, fail-first heuristics
- The MRV heuristic usually performs better than a random or static ordering, sometimes by orders of magnitude, although the results vary depending on the problem.



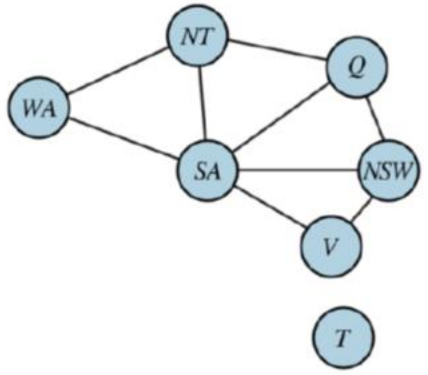
(a)

Ordering: variables

- MRV does not help in choosing the first region
- Degree heuristics
  - Choose the one with largest degrees
  - Attempts to reduce the branching factor
    - SA with degree 5 is chosen
  - Can also work as a tie-breaker when having same MRV

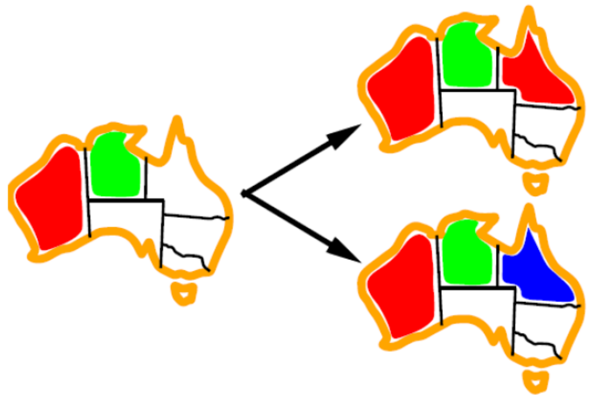


(a)



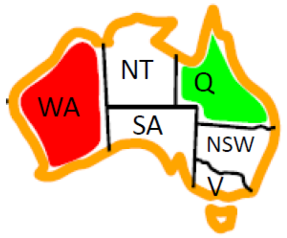
Ordering: values

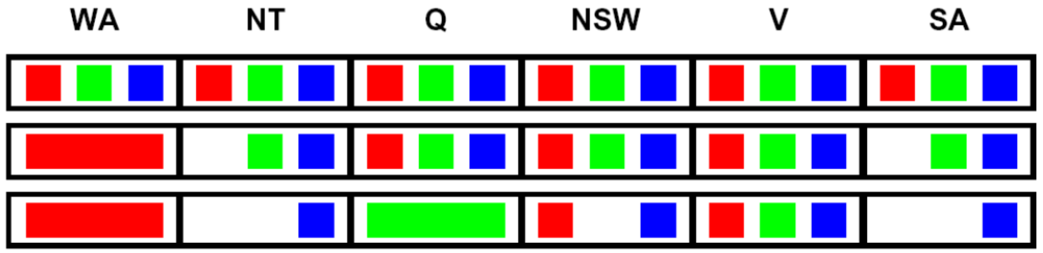
- Least-constraining-value (LCV)
- When we assign WA=Red and NT=Green
- Next choice is Q
  - Give Q red or blue?
  - LCV prefers red to blue
  - Leave the maximum flexibility for subsequent variable assignments



Filtering: forward checking

- Filtering: Keep track of domains for unassigned variables and cross off illegal values
- Forward checking: Cross off values that violate a constraint given existing assignment
- Constrain propagation
  - Using the constraints to reduce the number of legal values for a variable, which in turn reduce the legal values for another variable, and so on.



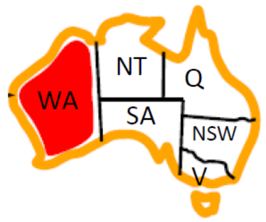


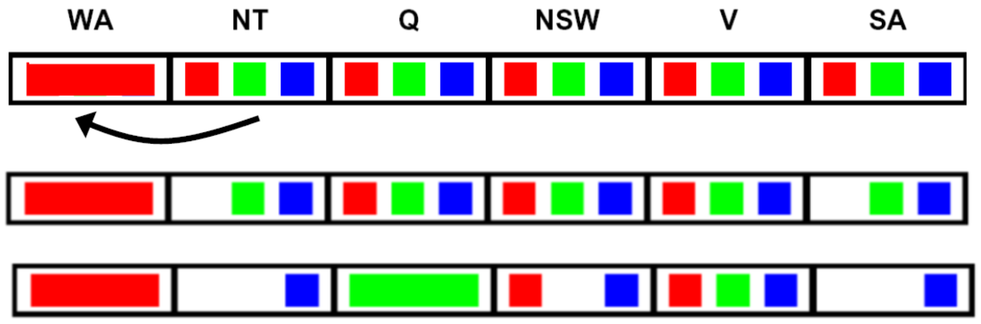
NT and SA cannot both be blue!

But we fail to detect this using constraint propagation.

Consistency of a single arc

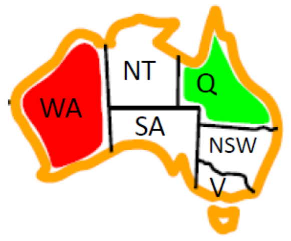
- An arc X →Y is consistent iff for *every* x in the tail X there is some y in the head Y which could be assigned without violating a constraint
- Tail = NT, head = WA
  - If NT = blue: we could assign WA = red
  - If NT = green: we could assign WA = red
  - If NT = red: there is no remaining assignment to WA that we can use
- Deleting NT = red from the tail makes this arc consistent
- Forward checking: Enforcing consistency of arcs pointing to each new assignment.

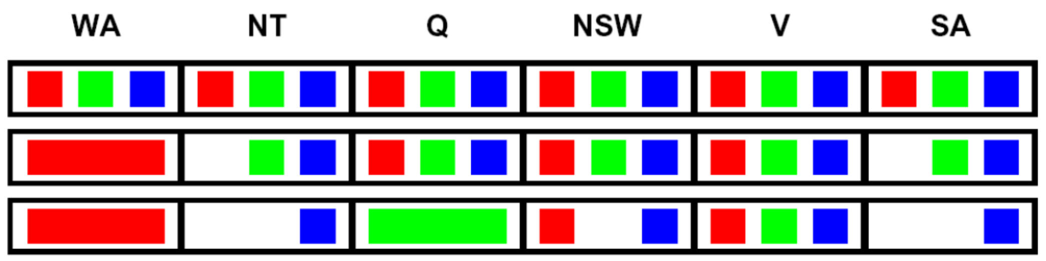




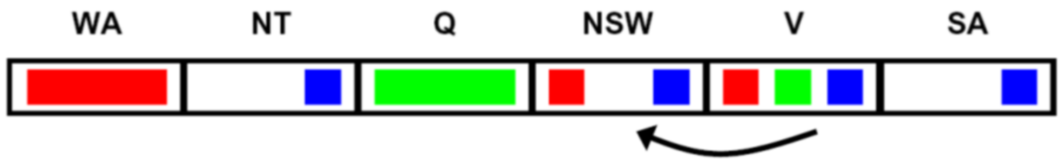
Arc Consistency of an entire CSP

- A simple form of propagation makes sure all arcs are consistent:
- After Q=Green
- Forward checking
  - NSW, SA, NT
  - No Green
- All arcs pointing to NSW, SA, and NT
- V → NSW
  - Every value in V, there is a legal value in NSW.



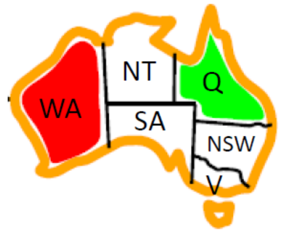


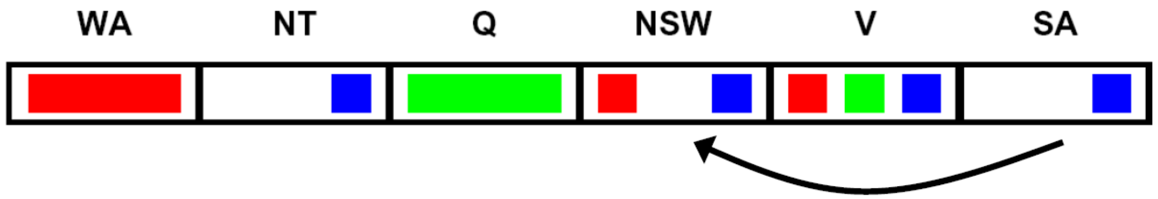
WA	NT	Q	NSW	V	SA
🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦
🟥	🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟩🟦
🟥	🟦	🟩	🟥🟦	🟥🟩🟦	🟦

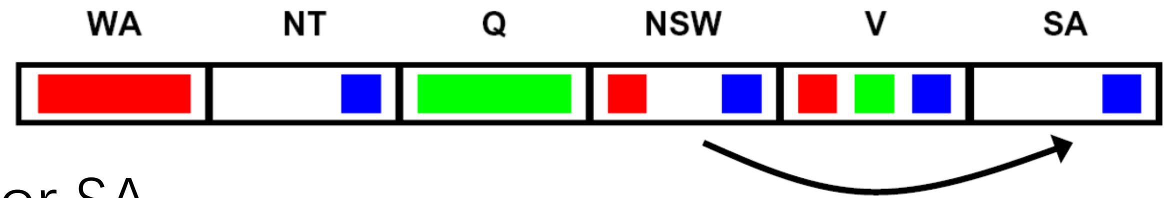


WA	NT	Q	NSW	V	SA
🟥	🟦	🟩	🟥🟦	🟥🟩🟦	🟦

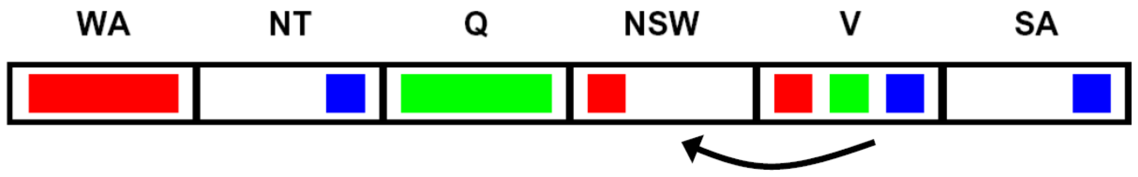
- All arcs pointing to NSW, SA, and NT
- V ⟶ NSW
  - OK
- SA ⟶ NSW
  - OK
- NSW ⟶SA
  - When NSW=Blue, no valid value for SA.
  - To make this arc consistent, delete NSW=Blue



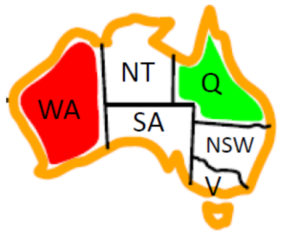


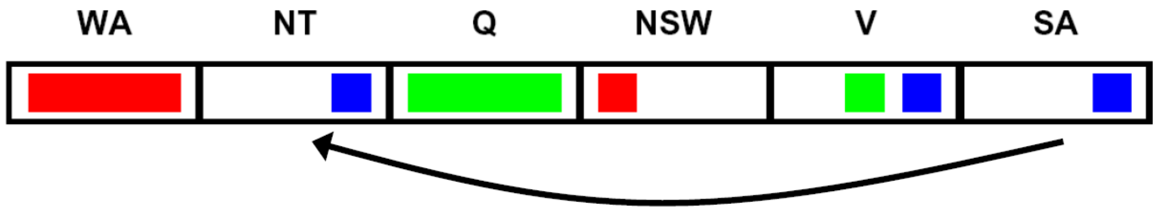


- Remember that arc V to NSW was consistent, when NSW had red and blue in its domain
- After removing blue from NSW, this arc might not be consistent anymore! We need to recheck this arc.
- Important: If X loses a value, neighbors of X need to be rechecked!
- $V \rightarrow NSW$
  - Red is deleted from V



- All arcs pointing to NSW, SA, and NT
- V → NSW
- SA → NSW
- NSW →SA
- V → NSW
- SA → NT
  - Remove blue from SA. SA becomes empty.
  - There is no way to solve this CSP with WA = red and Q = green, so we backtrack.
  - Arc consistency detects failure earlier than forward checking. Speed up.
  - Can be run as a preprocessor or after each assignment



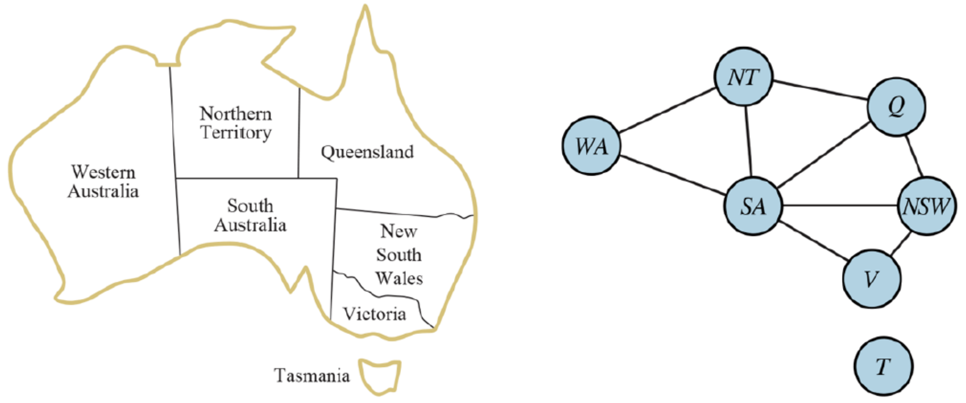


Outline

- Definition of CSPs
  - Variables, Domains, Constraints
- Types of constrains
  - Unary, binary, high-order, preferences
- Solving CSPs
  - Standard search
  - Backtracking
  - Problem structures
  - Local search

Problem structure

- Extreme case: independent subproblems
  - Example: Tasmania and mainland do not interact
- Independent subproblems are identifiable as connected components of constraint graph
- Constraint graph
  - Nodes corresponds to variables
  - Edge connects any two variables that participate a constraint



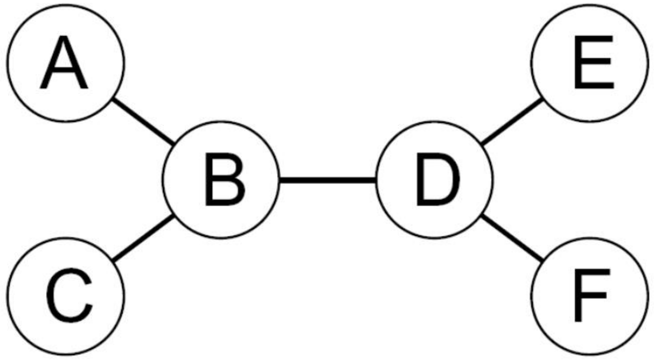
(a) (b)

(a) The principal states and territories of Australia. Coloring this map can be viewed as a constraint satisfaction problem (CSP). The goal is to assign colors to each region so that no neighboring regions have the same color. (b) The map-coloring problem represented as a constraint graph.

Independent subproblems

- Suppose a graph of n variables can be broken into subproblems of only c variables:
  - We then have n/c subproblems
  - Each subproblem is d^c
  - Worst-case solution cost is $O((n/c)(d^c))$, linear in n
  - For whole graph without subproblems, $O(d^n)$
- Example: $n = 80$, $d = 2$, $c = 20$
- 2^{80} = 4 billion years at 10 million nodes/sec
- $(4)(2^{20})$ = 0.4 seconds at 10 million nodes/sec

Tree-structure CSP

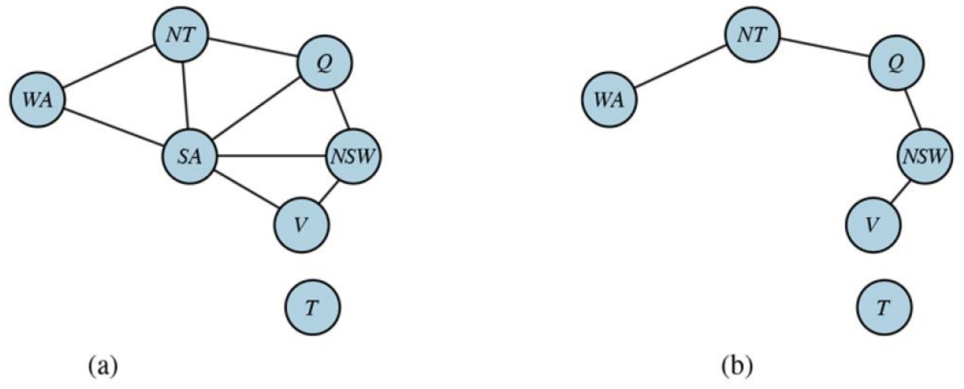


- Theorem: if the constraint graph has no loops, the CSP can be solved in $O(nd^2)$ time
    - Compare to general CSPs, where worst-case time is $O(d^n)$
- Choose any variable to be the root
- Choose an ordering of the variables
    - Topological sort
- Make this graph directed arc-consistent in $O(n)$ step
    - Each of which compare up to d possible domain values for two variables
    - Thus $O(nd^2)$

Reducing constraint graphs to trees

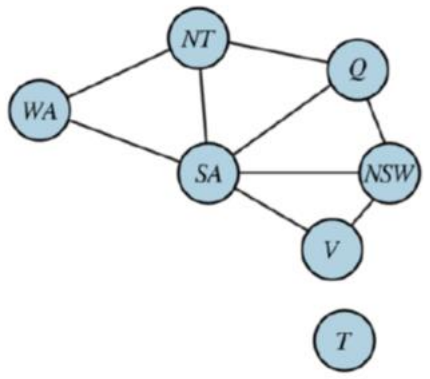
- Removing nodes
- The first way to reduce a constraint graph to a tree involves assigning values to some variables so that the remaining variables form a tree.
  - Fixing a value for SA
  - Deleting SA
  - Deleting from the domains of the other variables any values that are inconsistent with the value chosen for SA.

Figure 6.12



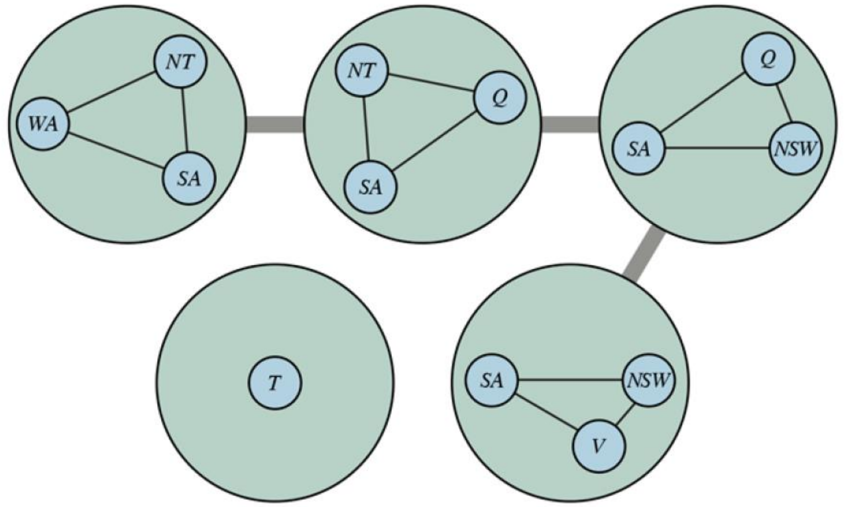
(a) The original constraint graph from Figure 6.1. (b) After the removal of SA, the constraint graph becomes a forest of two trees.

Reducing constraint graphs to trees



- Collapsing nodes together
- Tree decomposition
  - A transformation of the original graph into a tree where each node in the tree consists of a set of variables

Figure 6.13



A tree decomposition of the constraint graph in Figure 6.12(a).

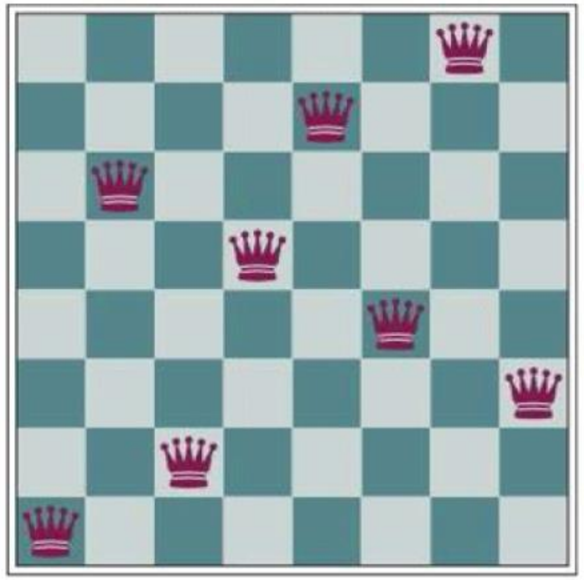
Outline

- Definition of CSPs
  - Variables, Domains, Constraints
- Types of constrains
  - Unary, binary, high-order, preferences
- Solving CSPs
  - Standard search
  - Backtracking
  - Problem structures
  - Local search

Local search for CSPs

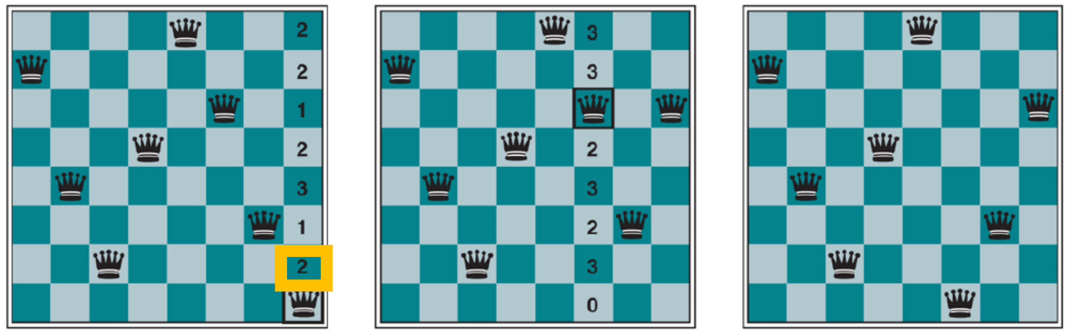
- Use a complete-state formulation
  - where each state assigns a value to every variable
- The search changes the value of one variable at a time.

- The 8-queens problem
  - Place 8 queens on a chess board so that no queen attacks another. (A queen attacks any piece in the same row, column, or diagonal.)

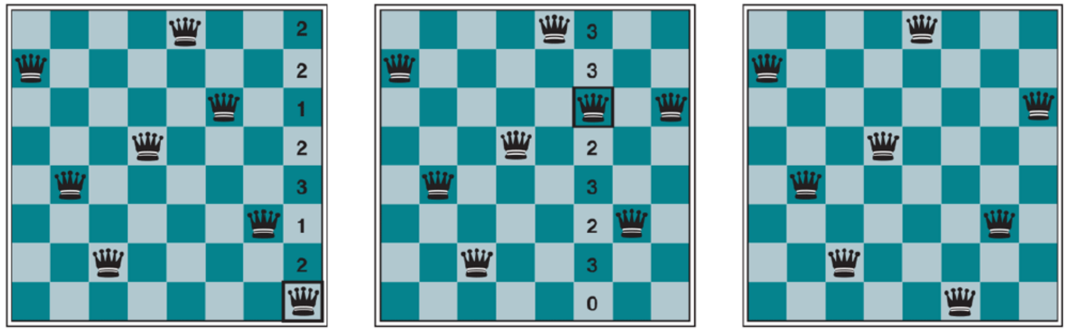


Iterative improvement

- Algorithm: While not solved,
  - Variable selection: randomly select any conflicted variable
  - Value selection: min-conflicts heuristic:
    - Choose a value that violates the fewest constraints
    - I.e., hill climb with h(n) = total number of violated constraints



- We start on the left with a complete assignment to the 8 variables; typically this will violate several constraints.
- We then randomly choose a conflicted variable, which turns out to be Q8, the rightmost column.
- We'd like to change the value to something that brings us closer to a solution; the most obvious approach is to select the value that results in the minimum number of conflicts with other variables
  - the min-conflicts heuristic
  - Why the highlight cell is 2: (Q8, Q3), (Q8, Q7)
- There are two positions with conflict number of 1
  - Q8 should moves to the 3rd or 6th row.



- Q6 conflicts with the new position of Q8
- Compute Q6’s minimum conflicting number
- Q6 moves to row 8.

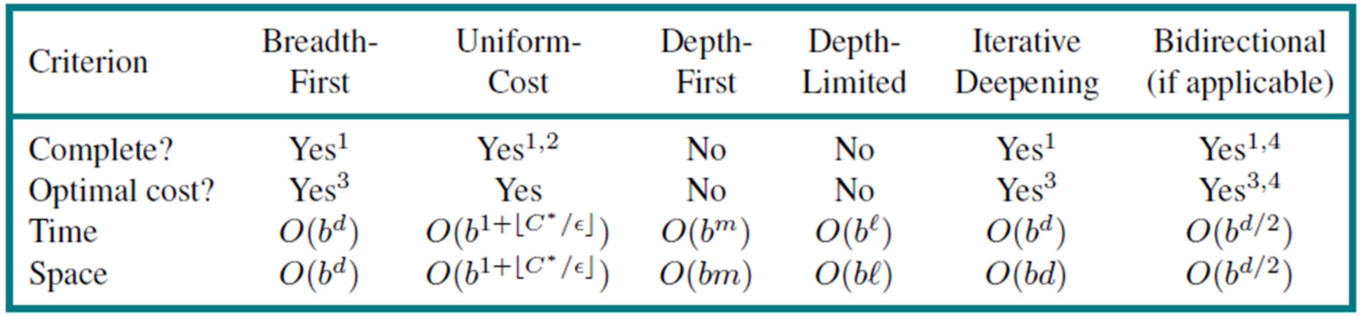
Lecture 6 ILOs

- Constraint Satisfaction Problems
  - Definition of CSPs
    - Variables, Domains, Constraints
  - Types of constrains
    - Unary, binary, high-order, preferences
  - Solving CSPs
    - Standard search
    - Backtracking
    - Problem structures
    - Local search

Local search for CSPs

- Min-conflicts is surprisingly effective for many CSPs.
- Amazingly, on the n-queens problem, if you don't count the initial placement of queens, the run time of min-conflicts is roughly independent of problem size.
  - It solves even the million-queens problem in an average of 50 steps (after the initial assignment).
- Benefits of local search
  - Online setting
  - Scheduling problem for an airline's weekly flights
  - Bad weather render current schedule infeasible.
  - Local search repairs the schedule with a minimum number of changes.
  - Backtracking search with new constraints may take much more time and may find solutions with many changes from the current schedule.

Lecture 2 ILOs



Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[1]	Yes[1,2]	No	No	Yes[1]	Yes[1,4]
Optimal cost?	Yes[3]	Yes	No	No	Yes[3]	Yes[3,4]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$

- Problem-solving agent
  - What is a problem-solving agent
  - Search problems and solutions
    - State space, initial state, goal states, action, transition model, action cost function
    - A solution: a path from the initial state to a goal state
- Example problems
  - Standardized problems
  - Real-world problems
- Search algorithms
  - Uninformed search vs. informed search
  - Performance measure: completeness, optimality, time complexity, space complexity
  - Best-First-Search: Which node from the frontier to expand next? Node with minimum f(n).
- Uninformed search
  - Breath-First-Search
  - Uniform cost search/Dijkstra's algorithm
  - Bidirectional search
  - Depth-First-Search
  - Iterative deepening search

Lecture 3 ILOs

A* search:	$g(n) + h(n)$	$(W = 1)$
Uniform-cost search:	$g(n)$	$(W = 0)$
Greedy best-first search:	$h(n)$	$(W = \infty)$
Weighted A* search:	$g(n) + W \times h(n)$	$(1 < W < \infty)$

- Informed Search Strategies
    - Heuristic function
    - Greedy best-first search
    - A* search, cost optimality, admissibility
    - Memory bounded search, weighted A* search
    - Design heuristics functions
        - Generating heuristics from relaxed problems
        - Generating heuristics from subproblems, pattern databases
        - Generating heuristics from with landmarks
        - Learning heuristics from experience
- Search in complex environments
    - Local search
        - Hill climbing
        - Simulated annealing
        - Local beam search
        - Evolutionary search

$$\mathbf{x} \leftarrow \mathbf{x} + \alpha \nabla f(\mathbf{x}) \,,$$

$$\mathbf{x} \leftarrow \mathbf{x} - \mathbf{H}_f^{-1}(\mathbf{x}) \nabla f(\mathbf{x}) \,,$$

Lecture 4 ILOs

- Search in complex environments
  - Local search in continuous spaces
    - Steepest ascent hill climbing, Newton method
  - Search with nondeterministic actions
    - AND-OR search, conditional plan
  - Search in partially observable environments
    - Sensorless problems
      - Belief-state space, ordinary search, sequences of actions in belief-state
    - Partially observable environments
      - AND-OR search, conditional plan
  - Online search agents and unknown environments

Lecture 5 ILOs

- Adversarial Search and Games
  - Game theory, problem settings
    - # players, deterministic vs. stochastic, zero-sum vs. general games
  - Optimal Decisions in games
    - binary outcome (win or lose): AND-OR tree search
    - Multiple outcomes: minimax search
    - Stochastic games: minimax search with chance nodes computing expected utilities
  - Alpha-Beta Pruning
  - Heuristic Alpha-Beta Tree Search
    - Cutoff, use heuristic evaluation
  - Monte Carlo Tree Search
    - Selection, Expansion, Simulation, and backpropagation
    - Exploration and exploitation