Intelligent Autonomous Agents
and Cognitive Robotics
Topic 3: Constraint Satisfaction Problems

Slides partly from Hwee Tou Ng's
Chapter 5 of AIMA

Outline

- Constraint Satisfaction Problems (CSP)
- Backtracking search for CSPs
- Multi-Agents → distributed backtracking

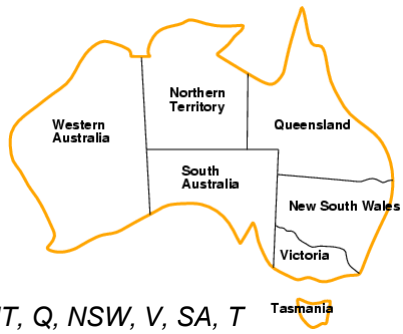
2

Constraint satisfaction problems (CSPs)

- Standard search problem:
  - state is a "black box“ – any data structure that supports successor function, heuristic function, and goal test
- CSP:
  - state is defined by variables X_i (i=1..n) with
  - values from domain D_i
  - goal test is a set of constraints C_m (m=1..z) 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

3

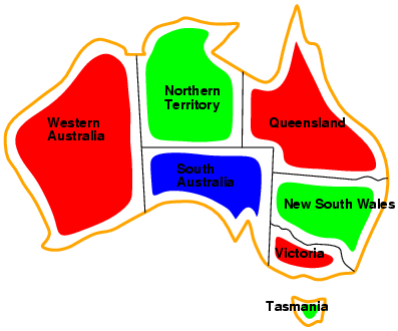
Visual example: Map-Coloring



- Variables: *WA, NT, Q, NSW, V, SA, T*
- Domains: $\forall i, D_i$ = {red, green, blue}
- Constraints: adjacent regions must have different colors
  - e.g., WA ≠ NT
  - or (WA,NT) ∈ {(red,green),(red,blue),(green,red), (green,blue), (blue,red), (blue,green)}

4

Example: Map-Coloring

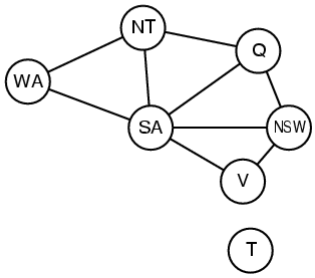


- Solutions are complete and consistent assignments, e.g.,
  - WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green

5

Constraint graph

- Binary CSP: each constraint relates two variables
- Constraint graph: nodes are variables, arcs are constraints



General-purpose CSP algorithms use the graph structure to speed up search. E.g., Tasmania is an independent sub problem!

6

Varieties of CSPs

- Discrete variables
  - finite domains:
    - n variables, domain size $d \rightarrow O(d^n)$ complete assignments
    - e.g., n-queens problem
  - infinite domains:
    - integers, strings, etc.
    - e.g., job scheduling, variables are start/end days for each job
    - need a constraint language, e.g., $StartJob_1 + 5 \leq StartJob_3$
- Continuous variables
  - e.g., start/end times for Hubble Space Telescope observations
  - linear constraints solvable in polynomial time by linear programming

7

Varieties of constraints

- Unary constraints involve a single variable,
  - e.g., SA ≠ green
- Binary constraints involve pairs of variables,
  - e.g., SA ≠ WA
- Higher-order constraints involve 3 or more variables

8

Real-world CSPs

- Assignment problems
  - e.g., who teaches what class
- Timetabling problems
  - e.g., which class is offered when and where; preferences
- Hardware configuration
- Transportation scheduling
- Factory scheduling

10

Constraint propagation

- In CSP an algorithm can do
  - Constraint propagation = inference
  - Search
  - Intertwined or as preprocessing
- The key idea is to create *local* consistency

11

Node consistency

- A variable is node-consistent if all the values satisfy the unary constraints
- Infer the values that are legal for a variable,
  - e.g. if South Australia does not like green, eliminate it {red, blue}
  - e.g. don't want to teach at 8 pm

12

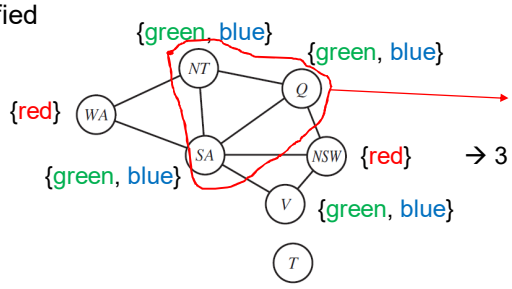
Global Constraints

- *Alldiff* (many algorithms)
  - Idea: If *m* variables have *n* values and *m>n* → can not be satisfied
    - Remove any variable with singleton domain and propagate this into other domains. Repeat as long as there are singleton domains.
    - If an empty domain is produced or m>n, then an inconsistency has been detected

13

Global Constraints

- *Alldiff* (many algorithms)
  - Idea: If *m* variables have *n* values and *m*>*n* → can not be satisfied



14

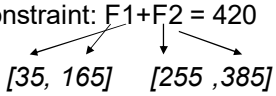
Resource Constraints

- Resource constraints: *Atmost*
  We can detect an inconsistency simply by checking the sum of the minimum values of the current domains:
  Atmost(10, P1, P2, P3, P4) persons for tasks.
  - Each variable has domain {3, 4, 5, 6}
    → can not be satisfied
  - Each variable has domain {2, 3, 4, 5, 6}
    → delete 5 and 6

15

Resource Constraints

- Bounds propagation/bounds consistent
  - In complex problems often not possible to enumerate domain values
  - Constraints:
    - Plane capacities for F1=[0, 165] , F2[0, 385]
    - Constraint: F1+F2 = 420



16

Resource Constraints

- Bounds propagation/bounds consistent
  - In complex problems often not possible to enumerate domain values
  - Constraints:
    - Plane capacities for F1=[0, 165] , F2[0, 385]
    - Constraint: F1+F2 = 420

→ *F1[35, 165] and F2[255, 385]*

  - We say that a CSP is **bounds consistent** if for every variable *X*, and for both the lower-bound and upper-bound values of *X*, there exists some value of *Y* that satisfies the constraint between *X* and *Y* for every variable *Y* . (Often used in praxis)

17

Standard search formulation

Let's start with the straightforward approach, then fix it

States are defined by the values assigned so far

- Initial state: the empty assignment { }
- Successor function: assign a value to an unassigned variable that does not conflict with current assignment
  → fail if no legal assignments
- Goal test: the current assignment is complete

1. Every solution appears at depth ***n*** with *n* variables
   → use depth-first search
2. Path is irrelevant
3. At the root we have n variables and d values b= nd
4. At depth ℓ we have b = (n - ℓ)d
5. All combinations n! · d^n leaves

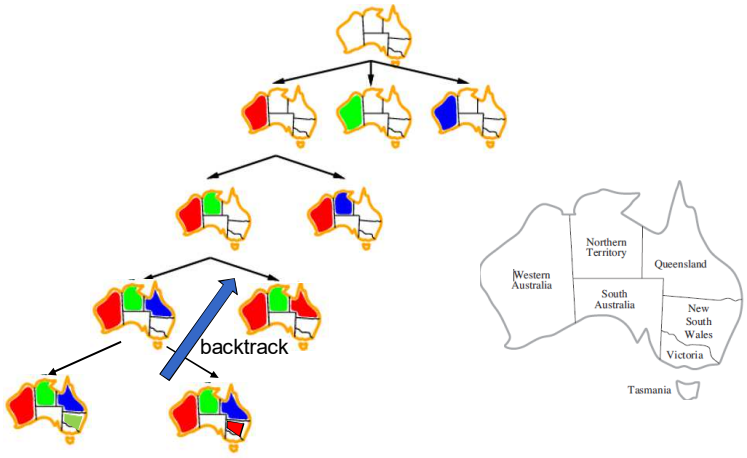
18

Backtracking search

- Variable assignments are commutative
  [WA = red then NT = green] same as [NT = green then WA = red]
- Only need to consider assignments to a single variable at each node
  → *b* = *d* branching factor, *n* variables → d^n leaves
- Depth-first search for CSPs with single-variable assignments is called backtracking search
- Backtracking search is the basic uninformed algorithm for CSPs

19

Backtracking example



20

Backtracking search

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

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

21

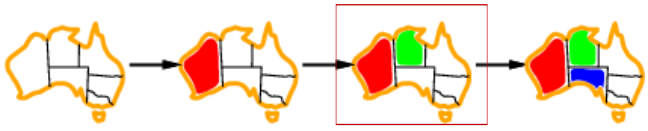
Improving backtracking efficiency

- General-purpose methods can give huge gains in speed:
  - Which variable should be assigned next SELECT-UNASSIGNED-VARIABLE?
  - In what order should its values be tried ORDER-DOMAIN-VALUES?
  - What inferences should be performed at each step in the search INFERENCE?
  - Can we detect inevitable failure early?

22

Most constrained variable

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



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

23

Most constrained variable

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

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

What about the first state

MRV does not help in the first state

24

Degree heuristic

- Tie-breaker among most constrained variables: Degree heuristic
- Most constraining variable:
  - choose the variable with the most constraints on remaining variables
  - used together with MRV

25

Least constraining value

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

- Combining these heuristics makes 1000 queens feasible

26

Inference: Forward checking

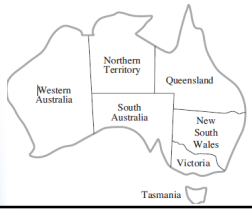
- Idea:
  - Keep track of remaining legal values for unassigned neighbors
  - Terminate search when any variable has no legal values

WA = red

28

Inference: Forward checking

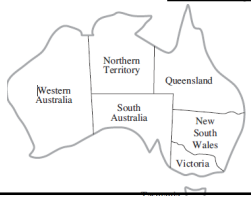
- Idea:
  - Keep track of remaining legal values for unassigned neighbors
  - Terminate search when any variable has no legal values



29

Inference: Forward checking

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



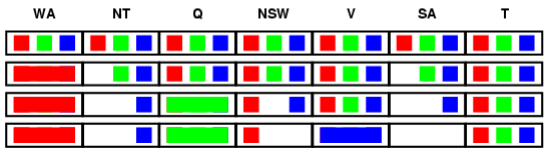
Q = green

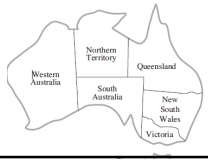
Victoria = blue

30

Inference: Forward checking

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





Victoria = blue

31

Forward checking

- 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, to neighbors

32

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 of Y

- Constraint $Y=X^2$ and domain {0,1,..9}. Can write the constraint as
  [(X, Y), {(0, 0), (1, 1), (2, 4), (3, 9))}]
  Can reduce the domains
  X = {0, 1, 2, 3}
  Y = {0, 1, 4, 9}

- What about (SA ≠ WA) and domain {red, green, blue}

  [(SA, WA),
  {(red , green), (red , blue), (green, red), (green, blue), (blue, red), (blue, green)}]

33

Arc consistency algorithm AC-3

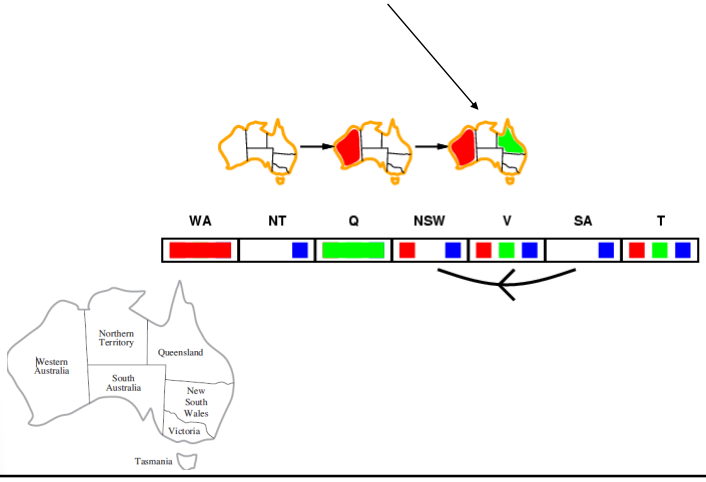
```
function AC-3( csp) returns the CSP, possibly with reduced domains
  inputs: csp, a binary CSP with variables {X_1, X_2, ..., X_n}
  local variables: queue, a queue of arcs, initially all the arcs in csp

  while queue is not empty do
    (X_i, X_j) ← REMOVE-FIRST(queue)
    if 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 ← 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 ↔ X_j
      then delete x from DOMAIN[X_i]; removed ← true
  return removed
```

- Time complexity: $O(cd^3)$

34

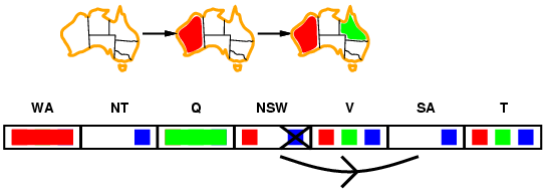
Arc consistency

- Assume we begin in state



35

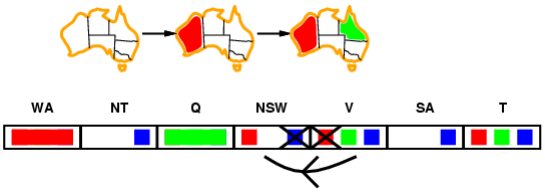
Arc consistency



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

36

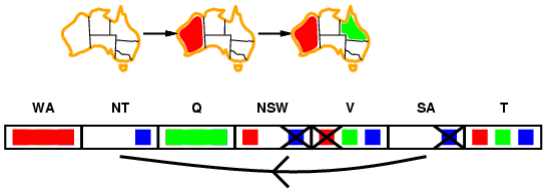
Arc consistency



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

37

Arc consistency



- If *X* loses a value, neighbors of *X* need to be rechecked
- Is run as a preprocessor
- Can also be modified to work with backtracking
  - On assignment put only (X_i, X_j) in the queue

38

Path consistency

- $\{X_i, X_j\}$ is path consistent with respect to X_m if for every consistent assignment
  there is an for X_m that is consistent.
  $\{X_i, X_m\}$ and $\{X_m, X_j\}$.
  See the CSP graph for detecting paths
- Could also be extended to K-Consistency

39

Multi-Agents CSP

- Also called distributed CSP
  - Variable and domain definition as before
  - Each agent owns a variable (many can be mapped to one)
  - Agents decides on value with relative autonomy
  - Has no global view on all dependencies
  - BUT! Can communicate with his neighbors in the constraint graph
- Many algorithms!! We only sketch one important algorithm

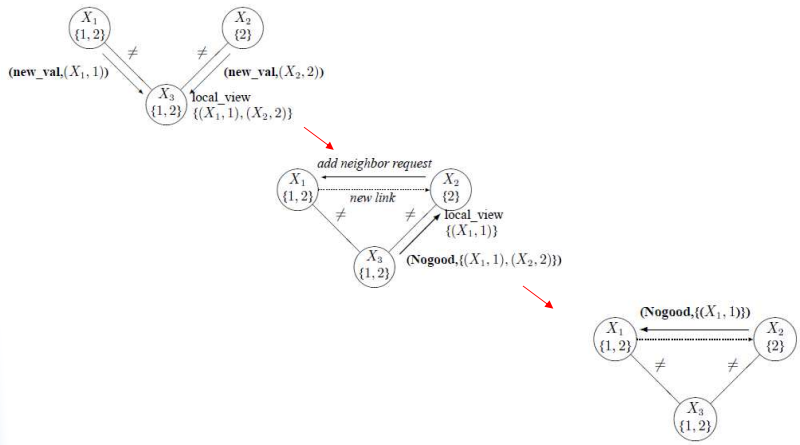
40

Multi-Agents CSP: Asynchronous Backtracking

- The algorithm makes an ordering on agents and assigns them priority numbers. All agents set their initial value concurrently
- a higher-priority agent *j* informs all lower-priority agents k_i of its assignment if connected in constraint graph
- lower-priority agent *k* evaluates the shared C_{jk} constraint with its own assignment
  - if constraints are satisfied with the current assignment → no action
  - otherwise, agent k looks for a different value consistent with choice of agent j
  - if such a consistent value exists → agent j adopts this value and informs other low-priority agents
  - if such a consistent value does not exist, agent j *updates NoGood list* and sends the message to agent j and seek for a value that is consistent with all connected higher priority agents
  - j receives a NoGood mentioning i it is not connected with j. j asks i to set up a link

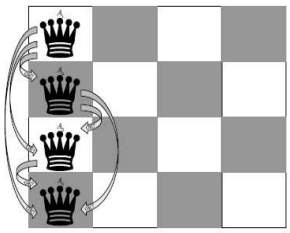
41

Adding edges



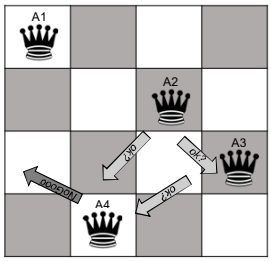
43

Example: 4-Queens



A1 knows no position
A2 knows A1
A3 knows A2 and A1
A4 knows all positions

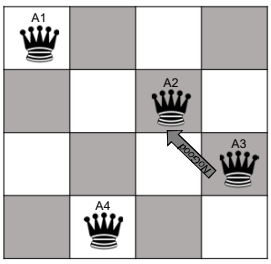
Based on local information each queen checks where to move or to resolve conflicts with upper queen. Afterwards do nothing, send “OK?” or “NoGood” messages.



NoGood: A1=1 and A2=1 → A3 ≠ 1

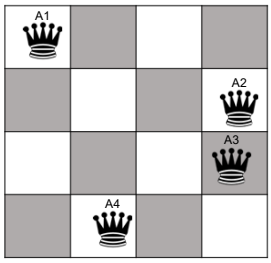
44

Example: 4-Queens



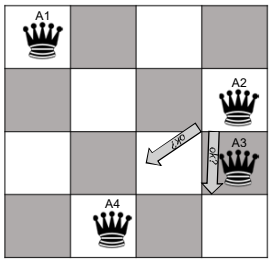
NoGood: A1=1 → A2 ≠ 3

Only A3 is active

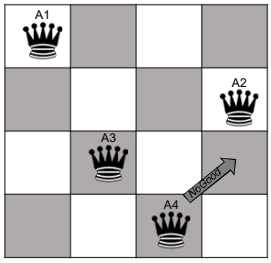


45

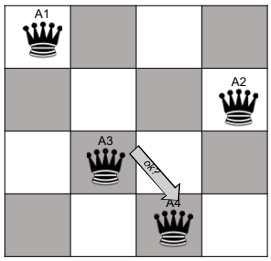
Example: 4-Queens



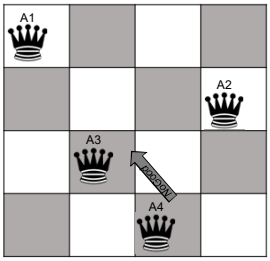
A4 sends a NoGood message:
A1=1 and A2=4 → A3 ≠ 4 (no longer valid)
and moves.



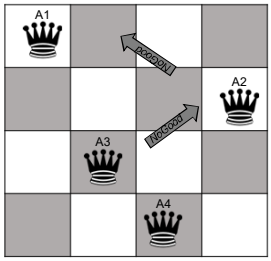
Example: 4-Queens

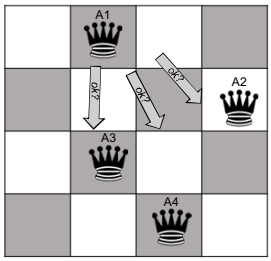


A4 sends a NoGood message:
A1=1 and A2=4 → A3 ≠ 2
and does not move, no conflict.



Example: 4-Queens



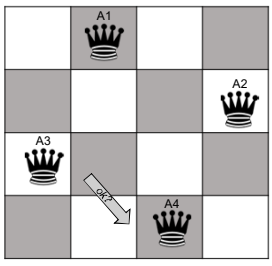


A3 has no option →NoGood: A1=1 → A2 ≠ 4,

A2 had a former NoGood message from A3 not to stay in 3
→send NoGood: A1 ≠ 1

48

Example: 4-Queens



No conflict for any queen → solved

49