CS 188: Artificial Intelligence

Constraint Satisfaction Problems II

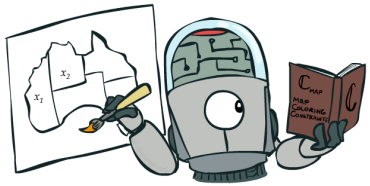
Instructor: Marco Alvarez

University of Rhode Island

[These slides were created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley.
All CS188 materials are available at http://ai.berkeley.edu.]

Constraint Satisfaction Problems

N variables
domain D
constraints



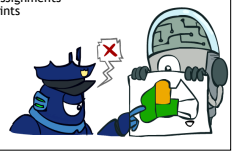
states — *partial assignment*

goal test — *complete; satisfies constraints*

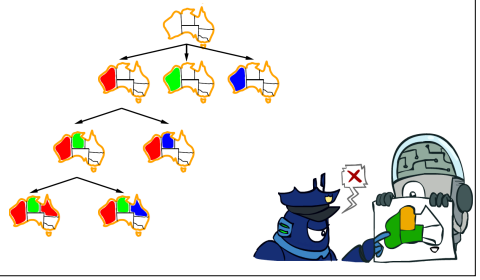
successor function — *assign an unassigned variable*

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 ≈ 25



Backtracking Example



Backtracking Search

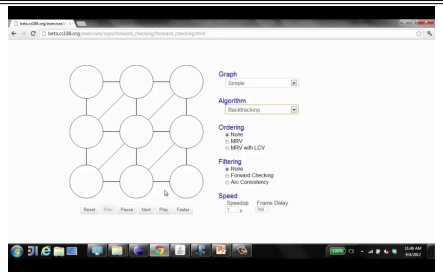
```
function BACKTRACKING-SEARCH(csp) returns solution/failure
  return RECURSIVE-BACKTRACKING({}, csp)

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
```

- Backtracking = DFS + variable-ordering + fail-on-violation

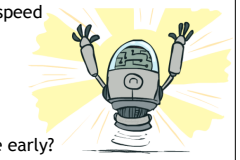
[Demo: coloring -- backtracking]

Video of Demo Coloring – Backtracking



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?

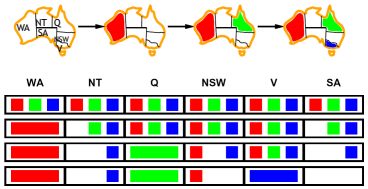


Filtering

Keep track of domains for unassigned variables and cross off bad options

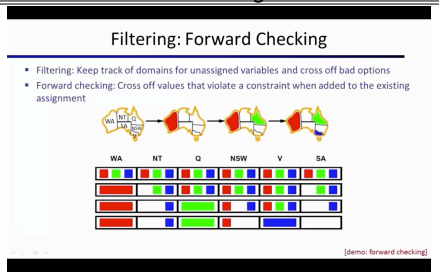
Filtering: Forward Checking

- Filtering: Keep track of domains for unassigned variables and cross off bad options
- Forward checking: Cross off values that violate a constraint when added to the existing assignment



[Demo: coloring -- forward checking]

Video of Demo Coloring – Backtracking with Forward Checking



10

Filtering: Constraint Propagation

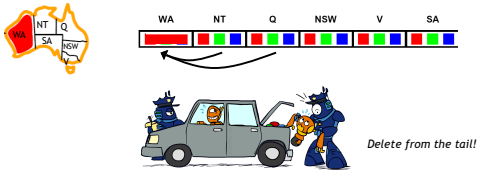
- Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures:

- NT and SA cannot both be blue!
- Why didn't we detect this yet?
- *Constraint propagation:* reason from constraint to constraint

11

Consistency of A Single Arc

- An arc X → Y is consistent iff for *every* x in the tail there is *some* y in the head which could be assigned without violating a constraint

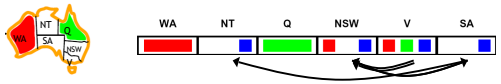


- Forward checking: Enforcing consistency of arcs pointing to each new assignment

12

Arc Consistency of an Entire CSP

- A simple form of propagation makes sure all arcs are consistent:



- Important: 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
- What's the downside of enforcing arc consistency?

Remember: Delete from the tail!

13

Enforcing Arc Consistency in a CSP

```
function AC-3(csp) returns the CSP, possibly with reduced domains
  inputs: csp, a binary CSP with variables {X1, X2, ..., Xn}
  local variables: queue, a queue of arcs, initially all the arcs in csp
  while queue is not empty do
    (Xi, Xj) ← REMOVE-FIRST(queue)
    if REMOVE-INCONSISTENT-VALUES(Xi, Xj) then
      for each Xk in NEIGHBORS[Xi] do
        add (Xk, Xi) to queue
---------------------------------------------------------------
function REMOVE-INCONSISTENT-VALUES(Xi, Xj) returns true iff succeeds
  removed ← false
  for each x in DOMAIN[Xi] do
    if no value y in DOMAIN[Xj] allows (x,y) to satisfy the constraint Xi ↔ Xj
      then delete x from DOMAIN[Xi]; removed ← true
  return removed
```

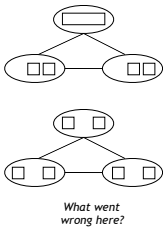
- Runtime: $O(n^2d^3)$, can be reduced to $O(n^2d^2)$
- … but detecting all possible future problems is NP-hard – why?

[Demo: CSP applet (made available by aispace.org) -- n-queens

14

Limitations of Arc Consistency

- After enforcing arc consistency:
  - Can have one solution left
  - Can have multiple solutions left
  - Can have no solutions left (and not know it)
- Arc consistency still runs inside a backtracking search!

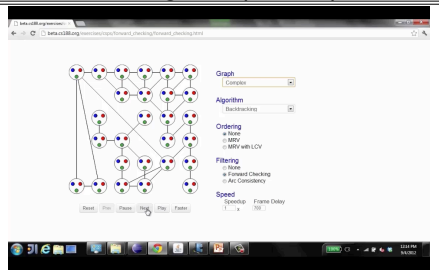


What went wrong here?

[Demo: coloring -- forward checking]
[Demo: coloring -- arc consistency]

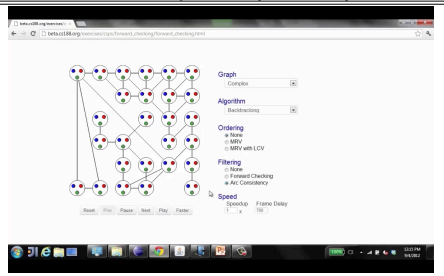
15

Video of Demo Coloring – Backtracking with Forward Checking – Complex Graph



16

Video of Demo Coloring – Backtracking with Arc Consistency – Complex Graph



17