CSE 473: Artificial Intelligence

Constraint Satisfaction

Daniel Weld

Slides adapted from Dan Klein, Stuart Russell, Andrew Moore & Luke Zettlemoyer

Space of Search Strategies

- Blind Search
  - DFS, BFS, IDS
- Informed Search
  - Systematic: Uniform cost, greedy, A*, IDA*
  - Stochastic: Hill climbing w/ random walk & restarts
- **Constraint Satisfaction**
  - Backtracking=DFS, FC, k-consistency
- Adversary Search

2

Recap: Search Problem

- States
  - configurations of the world
- Successor function:
  - function from states to lists of triples
    (state, action, cost)
- Start state
- Goal test

Recap: Constraint Satisfaction

- Kind of ***search*** in which
  - States are ***factored*** into sets of variables
  - Search = assigning values to these variables
  - Goal test is encoded with constraints
    - → Gives ***structure*** to search space
    - Exploration of one part informs others
- Special techniques add speed
  - Propagation
  - Variable ordering
  - Preprocessing

4

Constraint Satisfaction Problems

- Subset of search problems
- State is defined by
  - Variables X_i with values from a
  - Domain D (often D depends on i)
- Goal test is a set of constraints

Real-World CSPs

- Assignment problems: e.g., who teaches what class
- Timetabling problems: e.g., which class is offered when and where?
- Hardware configuration
- Gate assignment in airports
- Transportation scheduling
- Factory scheduling
- Fault diagnosis
- … lots more!
- Many real-world problems involve real-valued variables…

Chinese Food, Family Style

- Suppose k people…
  - Variables & Domains?
  - Constraints?

Factoring States

- Model state's (independent) parts, e.g.

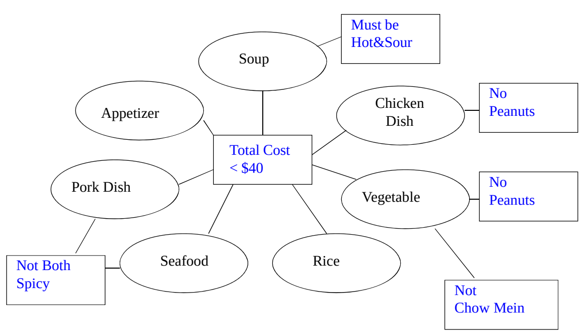
  Suppose every meal for n people

  Has n dishes plus soup
  - Soup =
  - Meal 1 =
  - Meal 2 =

    …
  - Meal n =

8

Chinese Constraint Network



9

Crossword Puzzle

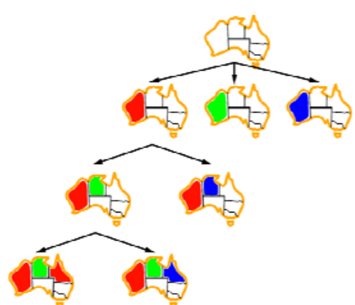
- Variables & domains?
- Constraints?

10

Standard Search Formulation

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

Backtracking Example



Backtracking Search

- Note 1: Only consider a single variable at each point
  - Variable assignments are commutative, so ***fix ordering of variables***
    - I.e., [WA = red then NT = blue] same as
      - [NT = blue then WA = red]
  - What is ***branching factor*** of this search?

Backtracking Search

Note 2: Only allow legal assignments at each point

- I.e. Ignore values which conflict previous assignments
- Might need some computation to eliminate such conflicts
- "Incremental goal test"

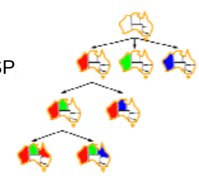
"Backtracking Search"

Depth-first search for CSPs with these two ideas

- One variable at a time, fixed order
- Only trying consistent assignments

Is called "Backtracking Search"

- Basic uninformed algorithm for CSP
- Can solve n-queens for n H 25



Backtracking Search

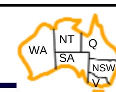
```
function BACKTRACKING-SEARCH(csp) returns solution/failure
   return RECURSIVE-BACKTRACKING({ }, csp)
function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
   if assignment is complete then return assignment
   var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
   for 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
```

- What are the choice points?

Improving Backtracking

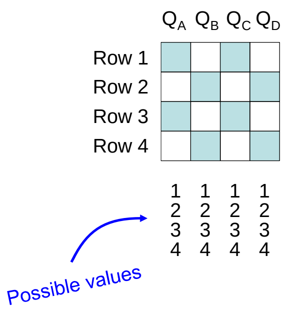
- General-purpose ideas give huge gains in speed
- Ordering:
  - Which variable should be assigned next?
  - In what order should its values be tried?
- Filtering: Can we detect inevitable failure early?
- Structure: Can we exploit the problem structure?

Forward Checking



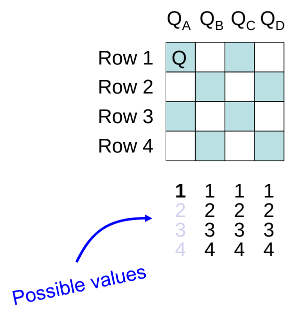
- Idea: Keep track of remaining legal values for unassigned variables (using immediate constraints)
- Idea: Terminate when any variable has no legal values

Forward Checking



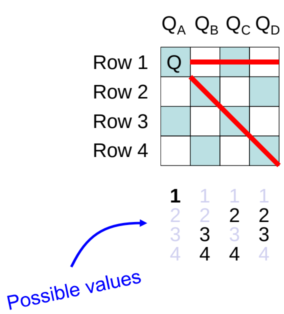
21

Forward Checking



22

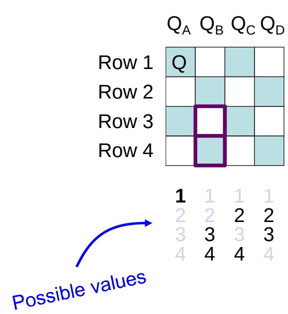
Forward Checking



Prune inconsistent values

23

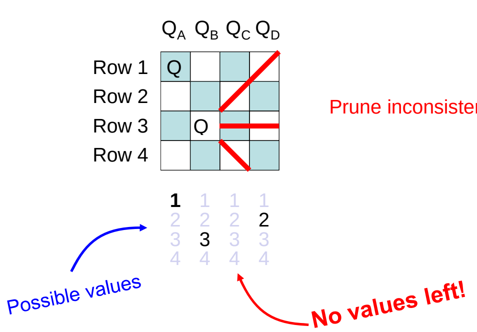
Forward Checking



Where can Q_B Go?

24

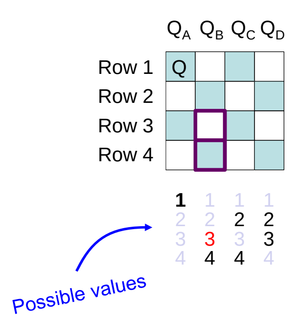
Forward Checking



Prune inconsistent values

25

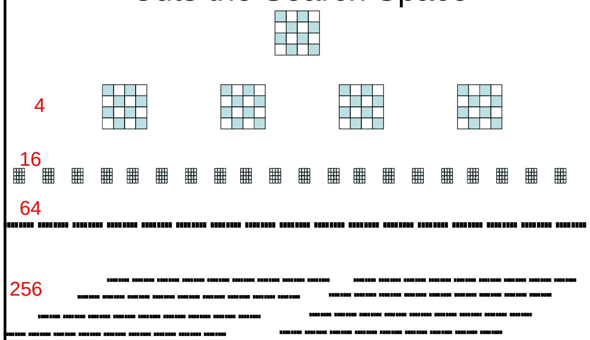
Forward Checking



Where can Q_B Go?

26

Forward Checking Cuts the Search Space

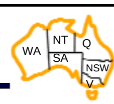


27

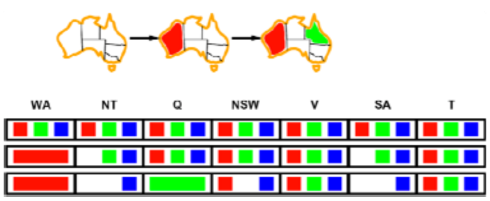
Are We Done?

28

Constraint Propagation

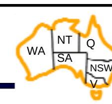


- Forward checking propagates information from assigned to adjacent unassigned variables, but doesn't detect more distant failures:

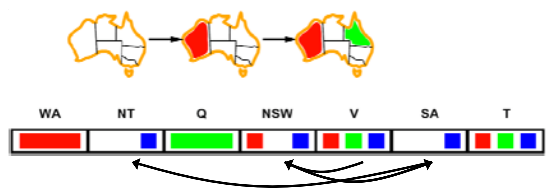


- NT and SA cannot both be blue!
- Why didn't we detect this yet?
- *Constraint propagation* repeatedly enforces constraints (locally)

Arc Consistency



- Simplest form of propagation makes each arc *consistent*
  - X □ Y is consistent iff for *every* value 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
- What's the downside of arc consistency?
- Can be run as a preprocessor or after each assignment

Arc Consistency

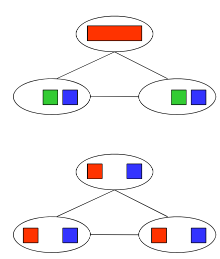
```
function AC-3(csp) returns the CSP, possibly with reduced domains
  inputs: csp, a binary CSP with variables {X1, X2, ..., Xn}
  local variables: queue, a queue of arcs, initially all the arcs in csp

  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: arc consistency animation]

Limitations of Arc Consistency

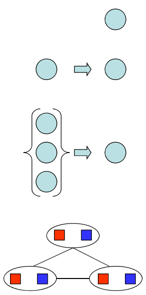
- After running arc consistency:
  - Can have one solution left
  - Can have multiple solutions left
  - Can have no solutions left (and not know it)



What went wrong here?

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
- (You need to know the k=2 algorithm)



Ordering: Minimum Remaining Values

- Minimum remaining values (MRV):
  - Choose the variable with the fewest legal values

- Why min rather than max?
- Also called "most constrained variable"
- "Fail-fast" ordering

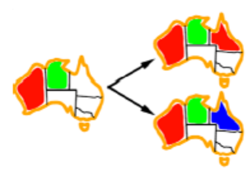
Ordering: Degree Heuristic

- Tie-breaker among MRV variables
- Degree heuristic:
  - Choose the variable participating in the most constraints on remaining variables

- Why most rather than fewest constraints?

Ordering: Least Constraining Value

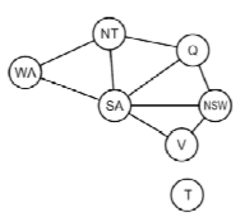
- Given a choice of variable:
  - Choose the *least constraining value*
  - The one that rules out the fewest values in the remaining variables
  - Note that it may take some computation to determine this!



- Why least rather than most?
- Combining these heuristics makes 1000 queens feasible

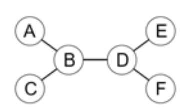
Problem Structure

- Tasmania and mainland are independent subproblems
- Identifiable as connected components of constraint graph
- Suppose each subproblem has c variables out of n total
- 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



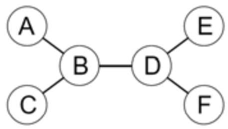
Tree-Structured CSPs

- Choose a variable as root, order variables from root to leaves such that every node's parent precedes it in the ordering



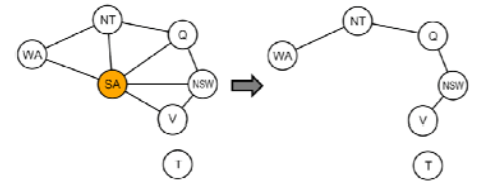
- For i = n : 2, apply RemoveInconsistent(Parent(X_i),X_i)
- For i = 1 : n, assign X_i consistently with Parent(X_i)
- Runtime: $O(n\ d^2)$

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 property also applies to logical and probabilistic reasoning: an important example of the relation between syntactic restrictions and the complexity of reasoning.

Nearly Tree-Structured CSPs

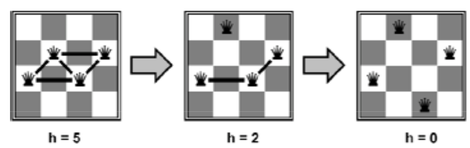


- Conditioning: instantiate a variable, prune its neighbors' domains
- Cutset conditioning: instantiate (in all ways) a set of variables such that the remaining constraint graph is a tree
- Cutset size c gives runtime $O((d^c)\ (n-c)\ d^2)$, very fast for small c

Iterative Algorithms for CSPs

- Greedy and local methods typically work with "complete" states, i.e., all variables assigned
- To apply to CSPs:
  - Allow states with unsatisfied constraints
  - Operators *reassign* variable values
- Variable selection: randomly select any conflicted variable
- Value selection by min-conflicts heuristic:
  - Choose value that violates the fewest constraints
  - I.e., hill climb with h(n) = total number of violated constraints

Example: 4-Queens

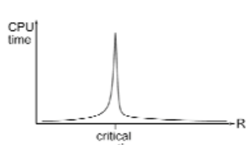


- States: 4 queens in 4 columns (4^4 = 256 states)
- Operators: move queen in column
- Goal test: no attacks
- Evaluation: h(n) = number of attacks

Performance of Min-Conflicts

- Given random initial state, can solve n-queens in almost constant time for arbitrary n with high probability (e.g., n = 10,000,000)
- The same appears to be true for any randomly-generated CSP *except* in a narrow range of the ratio

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



Summary

- CSPs are a special kind of search problem:
  - States defined by values (domains) of a fixed set of variables
  - Goal test defined by constraints on variable values
- Backtracking = DFS - one legal variable assigned per node
- Variable ordering and value selection heuristics help
- Forward checking prevents assignments that fail later
- Constraint propagation (e.g., arc consistency)
  - does additional work to constrain values and detect inconsistencies
- Constraint graph representation
  - Allows analysis of problem structure
- Tree-structured CSPs can be solved in linear time
- Iterative min-conflicts is usually effective in practice
  - Local (stochastic) search