CSE 573: 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**
- Adversary Search
  - Min-max, alpha-beta, expectimax, MDPS…

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

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 ***factored*** - defined by
  - Variables X_i with values from a
  - Domain D (often D depends on i)
- Goal test is a set of constraints

WHY STUDY?
- Simple example of a ***formal representation language***
- Allows more ***powerful search algorithms***

Example: Map-Coloring

- Variables: $WA,\ NT,\ Q,\ NSW,\ V,\ SA,\ T$
- Domain: $D = \{red, green, blue\}$
- Constraints: adjacent regions must have different colors

$$WA \neq NT$$

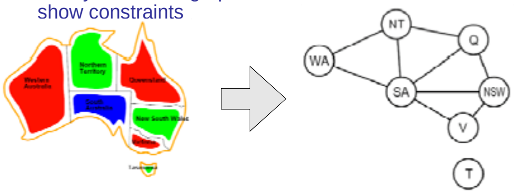
$$(WA, NT) \in \{(red, green), (red, blue), (green, red), \ldots\}$$

- Solutions are assignments satisfying all constraints, e.g.:

$$\{WA = red, NT = green, Q = red,$$
$$NSW = green, V = red, SA = blue, T = green\}$$

Constraint Graphs

- Binary CSP: each constraint relates (at most) two variables
- Binary constraint graph: nodes are variables, arcs show constraints



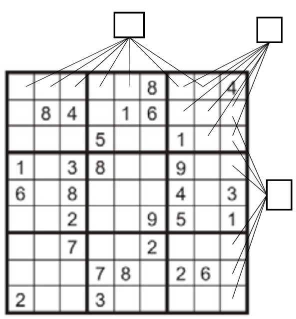
- General-purpose CSP algorithms use the graph structure to speed up search. E.g., Tasmania is an independent subproblem!

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…

Example: Sudoku



- Variables:
  - Each (open) square
- Domains:
  - {1,2,…,9}
- Constraints:

9-way alldiff for each column

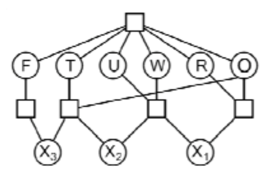
9-way alldiff for each row

9-way alldiff for each region

Example: Cryptarithmetic

- Variables (circles):
  $F\ T\ U\ W\ R\ O\ X_1\ X_2\ X_3$
- Domains:
  $\{0,1,2,3,4,5,6,7,8,9\}$
- Constraints (boxes):
  $\text{alldiff}(F,T,U,W,R,O)$
  $O + O = R + 10 \cdot X_1$
  $\cdots$

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



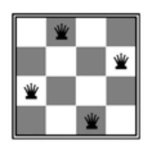
Crossword Puzzle

- Variables & domains?
- Constraints?

11

Example: N-Queens

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



$$\forall i,j,k \quad X_{ij} + X_{ik} \leq 1$$
$$\forall i,j,k \quad X_{ij} + X_{kj} \leq 1$$
$$\forall i,j,k \quad X_{ij} + X_{i+k,j+k} \leq 1$$
$$\forall i,j,k \quad X_{ij} + X_{i+k,j-k} \leq 1$$

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

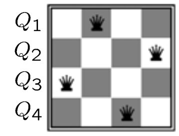
Example: N-Queens

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

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

Example: N-Queens

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



  - Constraints:

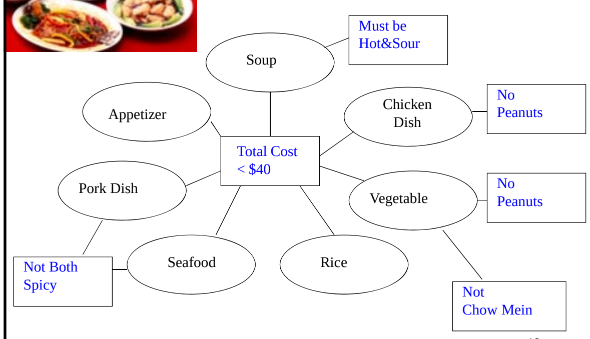
Implicit: $\forall i, j$ non-threatening(Q_i, Q_j)

-or-

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

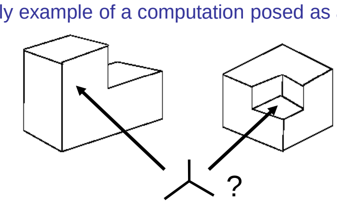
$\cdots$

Chinese Constraint Network



Example: The Waltz Algorithm

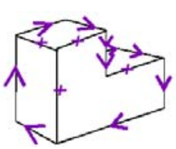
- The Waltz algorithm is for interpreting line drawings of solid polyhedra
- An early example of a computation posed as a CSP



- Look at all intersections
- Adjacent intersections impose constraints on each other

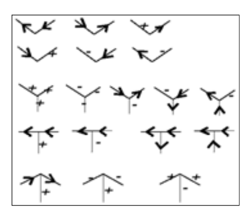
Waltz on Simple Scenes

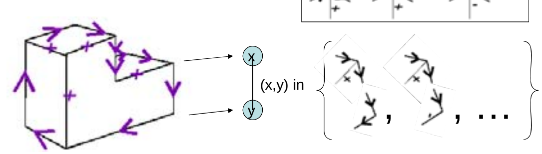
- Assume all objects:
  - Have no shadows or cracks
  - Three-faced vertices
  - "General position": no junctions change with small movements of the eye.
- Then each line on image is one of the following:
  - Boundary line (edge of an object) (>) with right hand of arrow denoting "solid" and left hand denoting "space"
  - Interior convex edge (+)
  - Interior concave edge (-)



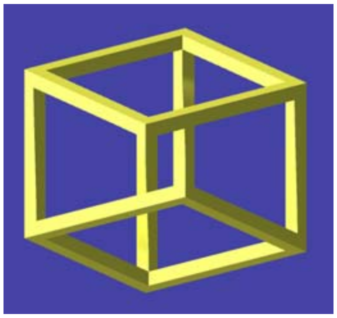
Legal Junctions

- Only certain junctions are physically possible
- How can we formulate a CSP to label an image?
- Variables: vertices
- Domains: junction labels
- Constraints: both ends of a line should have the same label





Local *vs* Global Consistency



22

Varieties of CSPs

- Discrete Variables
  - Finite domains
    - Size d means $O(d^n)$ complete assignments
    - E.g., Boolean CSPs, including Boolean satisfiability (NP-complete)
  - Infinite domains (integers, strings, etc.)
    - E.g., job scheduling, variables are start/end times for each job
    - Linear constraints solvable, nonlinear undecidable
- Continuous variables
  - E.g., start/end times for Hubble Telescope observations
  - Linear constraints solvable in polynomial time by LP methods

Varieties of Constraints

- Varieties of Constraints
  - Unary constraints involve a single variable (equiv. to shrinking domains):

    $$SA \neq green$$

  - Binary constraints involve pairs of variables:

    $$SA \neq WA$$

  - Higher-order constraints involve 3 or more variables:
    e.g., cryptarithmetic column constraints
- Preferences (soft constraints):
  - E.g., red is better than green
  - Often representable by a cost for each variable assignment
  - Gives constrained optimization problems
  - (We'll ignore these until we get to Bayes' nets)

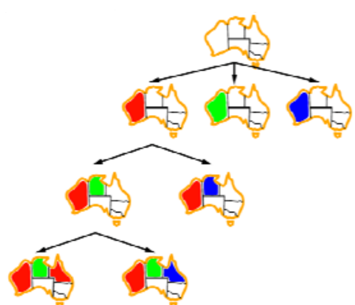
CSPs as Search?

- States?
- Successor function?
- Start state?
- Goal test?

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?

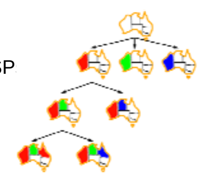
Backtracking Search

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 = 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
```

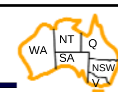
- What are the choice points?

Improving Backtracking

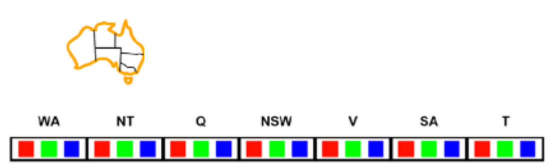
General-purpose ideas give huge gains in speed

- 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?

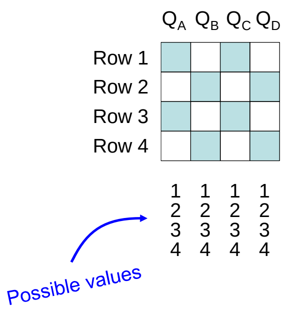
Forward Checking



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

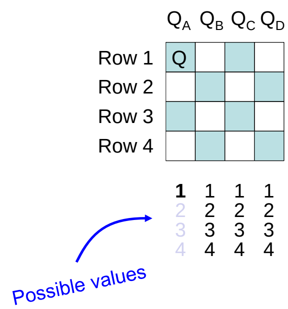


Forward Checking



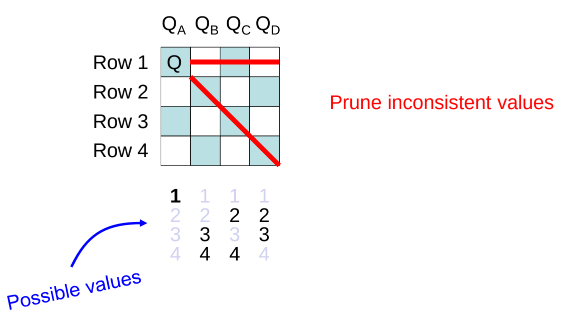
36

Forward Checking



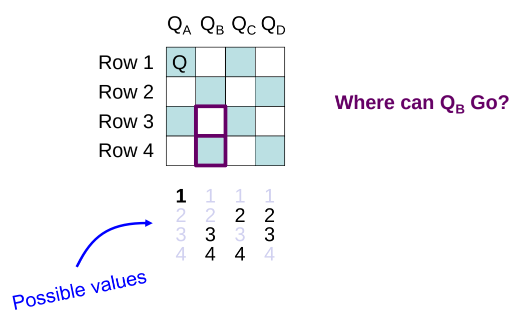
37

Forward Checking



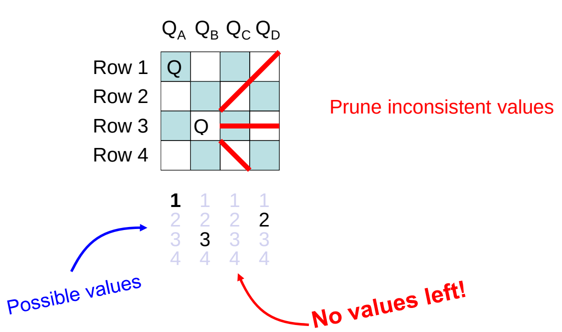
38

Forward Checking



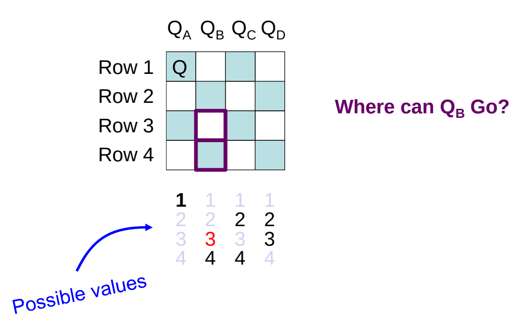
39

Forward Checking



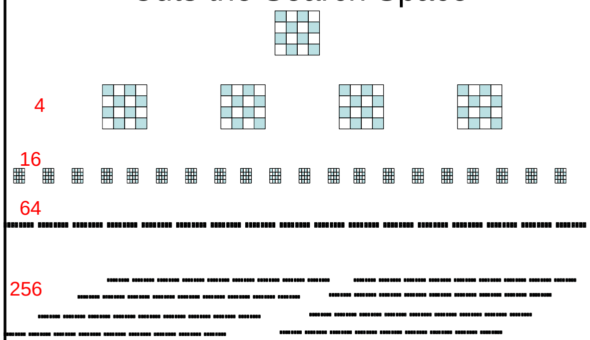
40

Forward Checking



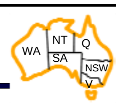
41

Forward Checking Cuts the Search Space



Are We Done?

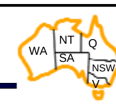
Constraint Propagation



- 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

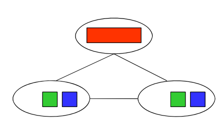
```
function AC-3( csp) returns the CSP, possibly with reduced domains
   inputs: csp, a binary CSP with variables {X1, X2, ..., Xn}
   local variables: queue, a queue of arcs, initially all the arcs in csp

   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
```

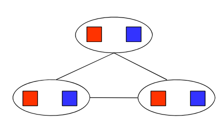
- Runtime: $O(n^2d^3)$, can be reduced to $O(n^2d^2)$
- … but detecting all possible future problems is NP-hard – why?

[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)

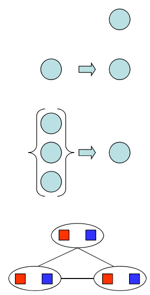




What went wrong here?

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



Variable Ordering Heuristics

- 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

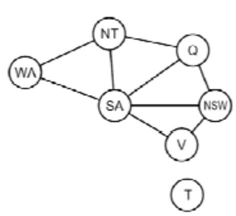
- Why most rather than fewest constraints?

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

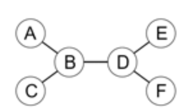
Problem Structure

- 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



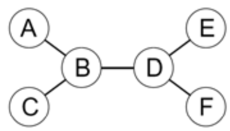
Tree-Structured CSPs

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



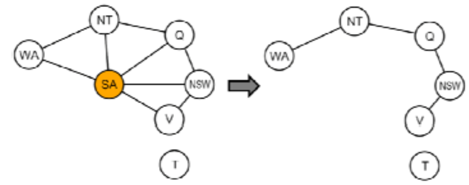
- For i = n : 2, apply RemoveInconsistent(Parent(X_i),X_i)
- For i = 1 : n, assign X_i consistently with Parent(X_i)
- Runtime: $O(n\ d^2)$

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

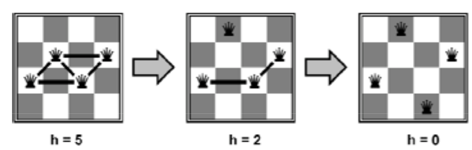


- Conditioning: instantiate a variable, prune its neighbors' domains
- Cutset conditioning: instantiate (in all ways) a set of variables such that the remaining constraint graph is a tree
- Cutset size c gives runtime O((d^c) (n-c) d^2), very fast for small c

Local Search for CSPs

- Greedy and stochastic methods typically search over “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 heuristic:
  - Min-conflicts
  - Choose value that violates the fewest constraints
  - I.e., hill climb with h(n) = total number of violated constraints

Example: 4-Queens

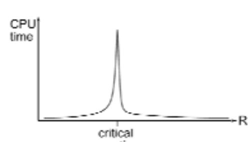


- States: 4 queens in 4 columns (4^4 = 256 states)
- Operators: move queen in column
- Goal test: no attacks
- Evaluation: h(n) = number of attacks

Performance of Min-Conflicts

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

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



CSP Summary

- CSPs are a special (factored) 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
- Local (stochastic) search often effective in practice
  - Iterative min-conflicts