CS 188: Artificial Intelligence
## Spring 2006

Lecture 6: CSPs
2/2/2006

Dan Klein – UC Berkeley

Many slides from either Stuart Russell or Andrew Moore


Constraint Satisfaction Problems

- Standard search problems:
  - State is a "black box": any old data structure
  - Goal test: any function over states
  - Successors: any map from states to sets of states

- Constraint satisfaction problems (CSPs):
  - State is defined by variables X_i with values from a domain D (sometimes D depends on i)
  - Goal test is a set of constraints 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
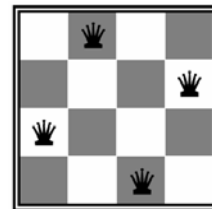
Example: N-Queens

- Formulation 1:
  - Variables: X_{ij}
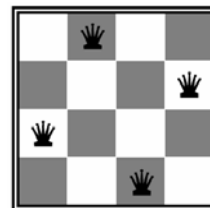 - Domains: $\{0, 1\}$
  - Constraints
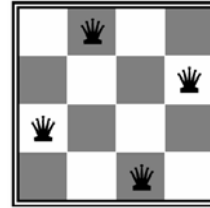
$$\forall i, j, k \quad (X_{ij}, X_{ik}) \in \{(0,0),(0,1),(1,0)\}$$
$$\forall i, j, k \quad (X_{ij}, X_{kj}) \in \{(0,0),(0,1),(1,0)\}$$
$$\forall i, j, k \quad (X_{ij}, X_{i+k,j+k}) \in \{(0,0),(0,1),(1,0)\}$$
$$\forall i, j, k \quad (X_{ij}, X_{i+k,j-k}) \in \{(0,0),(0,1),(1,0)\}$$
$$\sum_{i,j} X_{ij} = N$$

Example: N-Queens

- Formulation 2:
  - Variables: Q_k

  - Domains: $\{11, 12, 13, \ldots$
    $21, \ldots NN\}$

  - Constraints:

$$\forall i, j \quad (Q_i, Q_j) \in \{(11, 23), (11, 24), \ldots\}$$

$$\forall i, j \quad \text{non-threatening}(Q_i, Q_j)$$

Example: Map-Coloring

- Variables: $WA,\ NT,\ Q,\ NSW,\ V,\ SA,\ T$

- Domain: $D = \{red, green, blue\}$

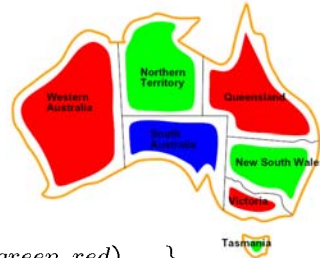- Constraints: adjacent regions must have different colors

    $WA \neq NT$

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

- 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

- 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!

3
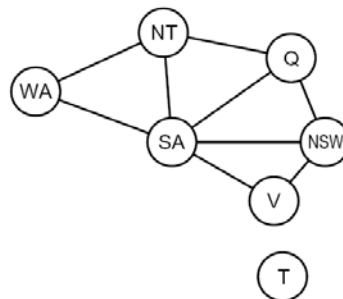
Example: Cryptarithmetic
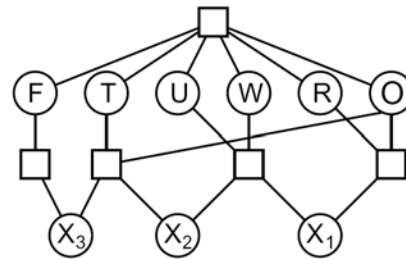
- Variables:

  $F\ T\ U\ W\ R\ O\ X_1\ X_2\ X_3$

- Domains:

  $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

- Constraints:

  $\text{alldiff}(F, T, U, W, R, O)$

  $O + O = R + 10 \cdot X_1$

  $\ldots$

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


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
    - Need a *constraint language*, e.g., $\text{StartJob}_1 + 5 < \text{StartJob}_3$
    - 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 (see cs170 for a bit of this theory)

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

Real-World CSPs

- Assignment problems: e.g., who teaches what class
- Timetabling problems: e.g., which class is offered when and where?
- Hardware configuration
- Spreadsheets
- Transportation scheduling
- Factory scheduling
- Floorplanning

- Many real-world problems involve real-valued variables…

Standard Search Formulation

- Standard search formulation of CSPs (incremental)

- Let's start with the straightforward, dumb 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
  - Goal test: the current assignment is complete and satisfies all constraints

Search Methods

- What does BFS do?

- What does DFS do?

- What's the obvious problem here?
- What's the slightly-less-obvious problem?
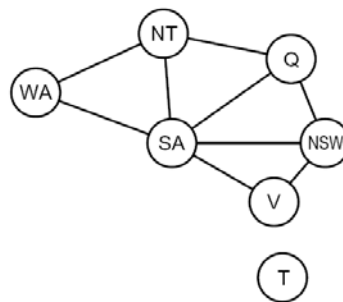
Backtracking Search

- Idea 1: Only consider a single variable at each point:
    - Variable assignments are commutative
    - 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
    - How many leaves are there?

- Idea 2: Only allow legal assignments at each point
    - I.e. consider only values which do not conflict previous assignmesnts

- Depth-first search for CSPs with these two improvements is called *backtracking search*

- Backtracking search is the basic uninformed algorithm for CSPs

- 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?

Backtracking Example

Improving Backtracking

- General-purpose ideas can give huge gains in speed:
  - Which variable should be assigned next?
  - In what order should its values be tried?
  - Can we detect inevitable failure early?
 - Can we take advantage of problem structure?

Minimum Remaining Values

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

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

Degree Heuristic

- Tie-breaker among MRV variables
- Degree heuristic:
    - Choose the variable with the most constraints on remaining variables
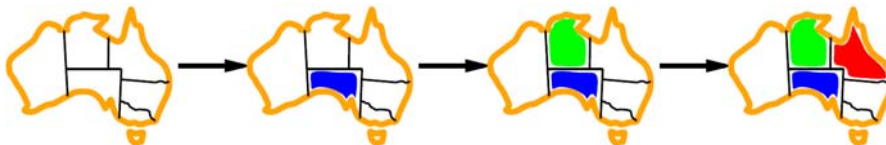
- Why most rather than fewest constraints?
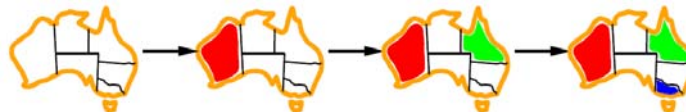
Least Constraining Value

- 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
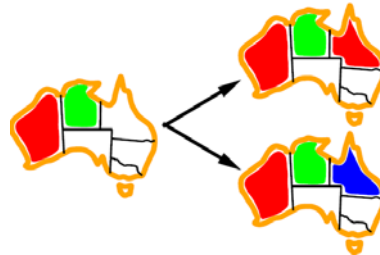
Forward Checking

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

WA	NT	Q	NSW	V	SA	T

Constraint Propagation

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

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

- 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*
  - $X \rightarrow Y$ is consistent iff for *every* value x there is *some* allowed y

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

- 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

11

Arc Consistency

```
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
```

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

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
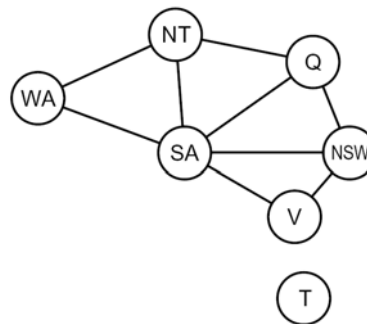
Tree-Structured CSPs
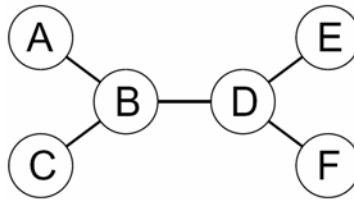
- Theorem: if the constraint graph has no loops, the CSP can be solved in $O(n\ d^2)$ time
  - Compare to general CSPs, where worst-case time is $O(d^n)$

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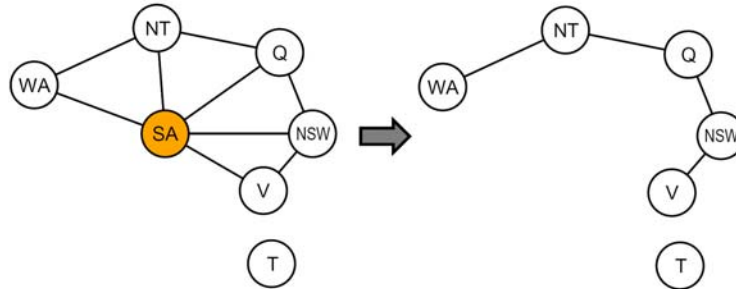

Tree-Structured CSPs

- Choose a variable as root, order variables from root to leaves such that every node's parent precedes it in the ordering
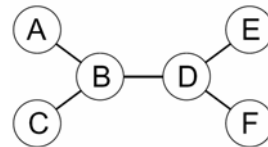
- For i = n : 2, apply RemoveInconsistent(Parent(X_i),X_i)
- For i = 1 : n, assign X_i consistently with Parent(X_i)

- Runtime: $O(n\ d^2)$

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 $O(\,(d^c)\,(n\text{-}c)\,d^2\,)$, very fast for small c

---

Iterative Algorithms for CSPs

- Hill-climbing, simulated annealing 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., hillclimb with h(n) = total number of violated constraints

Example: 4-Queens

h = 5   h = 2   h = 0
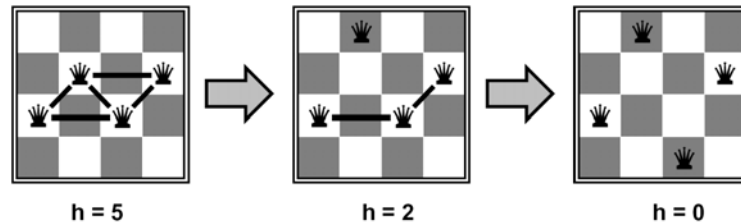
- States: 4 queens in 4 columns (4^4 = 256 states)
- Operators: move queen in column
- Goal test: no attacks
- Evaluation: h(n) = number of attacks

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

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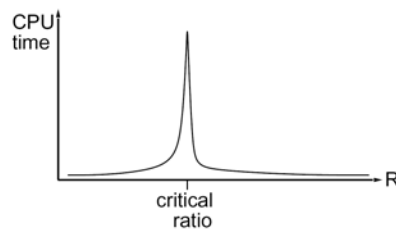
Summary

- CSPs are a special kind of search problem:
  - States defined by values of a fixed set of variables
  - Goal test defined by constraints on variable values

- Backtracking = depth-first search with one legal variable assigned per node

- Variable ordering and value selection heuristics help significantly

- Forward checking prevents assignments that guarantee later failure

- Constraint propagation (e.g., arc consistency) does additional work to constrain values and detect inconsistencies

- The 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