Announcements

- Project 1: Search is due next week
- Written 1: Search and CSPs out soon
- Piazza: check it out if you haven’t

CS 188: Artificial Intelligence
Fall 2011

Lecture 4: Constraint Satisfaction
9/6/2011

Dan Klein – UC Berkeley
Multiple slides adapted from Stuart Russell or Andrew Moore
What is Search For?

- Models of the world: single agents, deterministic actions, fully observed state, discrete state space

- Planning: sequences of actions
  - The path to the goal is the important thing
  - Paths have various costs, depths
  - Heuristics to guide, fringe to keep backups

- Identification: assignments to variables
  - The goal itself is important, not the path
  - All paths at the same depth (for some formulations)
  - CSPs are specialized for identification problems

Constraint Satisfaction Problems

- Standard search problems:
  - State is a “black box”: arbitrary data structure
  - Goal test: any function over states
  - Successor function can be anything

- Constraint satisfaction problems (CSPs):
  - A special subset of search problems
  - 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: Map-Coloring

- **Variables:** \( \text{WA}, \text{NT}, \text{Q}, \text{NSW}, \text{V}, \text{SA}, \text{T} \)
- **Domain:** \( D = \{ \text{red, green, blue} \} \)
- **Constraints:** adjacent regions must have different colors
  \[ \text{WA} \neq \text{NT} \]
  \[ (\text{WA}, \text{NT}) \in \{ (\text{red, green}), (\text{red, blue}), (\text{green, red}), \ldots \} \]
- **Solutions are assignments satisfying all constraints, e.g.:**
  \[ \{ \text{WA = red}, \text{NT = green}, \text{Q = red}, \text{NSW = green}, \text{V = red}, \text{SA = blue}, \text{T = green} \} \]

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: $\{1, 2, 3, \ldots N\}$

- **Constraints:**
  - Implicit: $\forall i, j \text{ non-threatening}(Q_i, Q_j)$
  - Explicit: $(Q_1, Q_2) \in \{(1, 3), (1, 4), \ldots\}$
    
  
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!
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):
  \[ \text{alldiff}(F, T, U, W, R, O) \]
  \[ O + O = R + 10 \cdot X_1 \]
  \[ \ldots \]

Example: Sudoku

- Variables:
  - Each (open) square
- Domains:
  - \( \{1, 2, \ldots, 9\} \)
- Constraints:
  - 9-way alldiff for each column
  - 9-way alldiff for each row
  - 9-way alldiff for each region
  (or can have a bunch of pairwise inequality constraints)
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

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
    (see cs170 for a bit of this theory)
Varieties of Constraints

- **Varieties of Constraints**
  - Unary constraints involve a single variable (equiv. to shrinking domains):
    \[ SA \neq \text{green} \]
  - 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)

Real-World CSPs

- Assignment problems: e.g., who teaches what class
- Timetabling problems: e.g., which class is offered when and where?
- Hardware configuration
- Transportation scheduling
- Factory scheduling
- Floorplanning
- Fault diagnosis
- … lots more!

- 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 defined by the values assigned so far (partial assignments)
  - 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
- Simplest CSP ever: two bits, constrained to be equal

Search Methods

- What would BFS do?

- What would DFS do?

- What problems does this approach have?
Backtracking Search

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

- Idea 2: Only allow legal assignments at each point
  - I.e. consider only values which do not conflict previous assignments
  - Might have to do some computation to figure out whether a value is ok
  - “Incremental goal test”

- Depth-first search for CSPs with these two improvements is called backtracking search (useless name, really)
  - [DEMO]

- Backtracking search is the basic uninformed algorithm for CSPs
- Can solve n-queens for n \approx 25

---

function Backtracking Search(esp) returns solution/failure
return Recursive-Backtracking({} esp)

function Recursive-Backtracking(assignment, esp) returns soln/failure
if assignment is complete then return assignment
var ← Select-Unassigned-Variable(Variables[esp], assignment, esp)
for each value in Order-Domain-Values(var, assignment, esp) do
  if value is consistent with assignment given CONSTRAINTS[esp] then
    add \{var = value\} to assignment
    result ← Recursive-Backtracking(assignment, esp)
    if result \neq failure then return result
    remove \{var = value\} from assignment
  return failure

- Backtracking = DFS + var-ordering + fail-on-violation
- What are the choice points?

[DEMO: backtracking]
Backtracking Example

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?
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: 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
Filtering: Forward Checking

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

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* propagates from constraint to constraint
Consistency of An Arc

- An arc \( X \rightarrow Y \) 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 each arc pointing to the new assignment.

Arc Consistency of a CSP

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

- 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 enforcing 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 \{X_1, X_2, \ldots, 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
```

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

[DEMO]

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?
Demo: Backtracking + AC