Announcements

- Project 1: Search is due Monday
  - Looking for partners? After class or newsgroup

- Written 1: Search and CSPs out soon

- Newsgroup: check it out

CS 188: Artificial Intelligence
Fall 2010

Lecture 4: Constraint Satisfaction
9/7/2010

Dan Klein – UC Berkeley
Multiple slides adapted from Stuart Russell or Andrew Moore
Today

- Search Conclusion
- Constraint Satisfaction Problems

A* Review

- A* uses both backward costs $g$ and forward estimate $h$: $f(n) = g(n) + h(n)$
- A* tree search is optimal with admissible heuristics (optimistic future cost estimates)
- Heuristic design is key: relaxed problems can help
**A* Graph Search Gone Wrong**

**State space graph**

- **S**: Initial state with $h=2$
- **A**: State with $h=4$, connected to **S** with cost 1
- **B**: State with $h=1$, connected to **A** with cost 1
- **C**: State with $h=1$, connected to **A** with cost 1 and to **B** with cost 2
- **G**: Goal state with $h=0$

**Search tree**

- **S (0+2)** to **A (1+4)**
- **A (1+4)** to **C (2+1)**
- **C (2+1)** to **G (5+0)**
- **B (1+1)** to **C (3+1)**
- **C (3+1)** to **G (6+0)**

**Consistency**

- **Definition:**
  \[ \text{cost}(A \text{ to } C) + h(C) \geq h(A) \]
- **Consequences:**
  - The f value along a path never decreases
  - Non-decreasing f means you’re optimal to every state (not just goals)
  - A* graph search is optimal
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: $WA, NT, Q, NSW, V, SA, T$
- 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.:

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

Example: N-Queens

- 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: $\forall i, j \text{ non-threatening}(Q_i, Q_j)$
  - 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, ..., 9\}

- **Constraints:**
  
  9-way alldiff for each column
  
  9-way alldiff for each row
  
  9-way alldiff for each region
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 $O(d^p)$ 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 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
- 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?

[demo: dfs]
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 ≈ 25

---

```python
function BACKTRACKING_SEARCH(esp) returns solution/failure
return RECURSIVE-BACKTRACKING(∅, esp)

function RECURSIVE-BACKTRACKING(assignment, esp) returns solution/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 ≠ 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?
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

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.

```
WA NT Q NSW V SA T
```

- 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:

```
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 enforcing arc consistency?
- Can be run as a preprocessor or after each assignment.
Arc Consistency

```plaintext
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 \text{Neighbours}[X_i] do
            add \((X_k, X_i)\) to queue

function REMOVE-INCONSISTENT-VALUES\((X_i, X_j)\) returns true iff succeeds
removal ← false
for each \(x\) in \text{Domain}[X_i] do
    if no value \(y\) in \text{Domain}[X_j] allows \((x, y)\) to satisfy the constraint \(X_i \leftrightarrow X_j\)
        then delete \(x\) from \text{Domain}[X_i]; removal ← true
return removal
```

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

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?