Introduction to Artificial Intelligence

Lecture 6 – CSPs (cont.)

CS/CNS/EE 154
Andreas Krause
Announcements

- Homework 1 is out. Due Friday Oct 22
- Room for recitation and office hours:
  - Annenberg 107; Tuesday and Thursday 4:30-5:30pm
- Project assignments have been sent out
- Will post details on evaluation soon

- “Science of Iron Man” tonight 8pm (Beckman Auditorium)
Constraint satisfaction problems

- So far: “black box search”
  - Environment state is arbitrary object

- CSPs:
  - state is defined by variables $X_i$ taking values in domain $D_i$
  - goal test is a set of constraints
  - step cost is 0 – just need to find goal
    (or prove that constraints can’t be satisfied)

- Can develop general purpose algorithms for large class of problems
Example: Map coloring

Variables:
WA, NT, SA, Q, NSW, V, T

Domains:
{R, B, Y3}

Constraints:
\[\text{WA} \neq \text{NT} \land \text{WA} \neq \text{SA} \land \ldots \]
\[= (\text{WA} \land \text{NT}) \land (R, Y), (R, B), (Y, B), (Y, R)\]

- Variables? Domains? Constraints?
Types of CSPs

- Discrete variables
  - Finite domains: Map Coloring, Sudoku, 8 queens, SAT
  - Infinite domains: 
    \[(x_2 \geq x_1 + 3) \land (x_5 \geq x_3 + 2)\]
    Job scheduling
    \(k_i\): start time of job \(i\), Domain: \([1, \infty)\)

- Continuous variables
  - Robot/factory control, time flexibility, ...
Types of constraints

- **Unary**: involve single variable  
  \[ NSW = 3 \]

- **Binary**: involve pairs of variables  
  \[ NSW \neq NT, \ldots \]

- **Higher-order**: involve 3 or more variables

- **Soft constraints**: violation incurs cost
  - Constraint optimization instead of satisfaction
Solving CSP with search

- Naïve approach
  - State = Partial assignment to variables
  - Successor fn = Assign feasible value to some unassigned var
  - Goal test = check constraints

- Problems?

  \[ \text{Size of search tree: } O(n! \cdot d^n) \]
Backtracking search

- Variable assignments are commutative!
  \[ N_{SW} = Y \text{ then } N_T = B \text{ same as } N_T = B \text{ then } N_{SW} = Y \]

- Only need to consider assignments to single variable at each node
  
  \[ \text{Size of tree: } O(d^n) \ll n!d^n \]

- Depth-first search with single var. assignments is called backtracking search

- Can solve 25-queens
Backtracking example
Backtracking example
Backtracking example
Backtracking example
Improving backtracking search

- General purpose methods can drastically improve speed

1. Which variable should be assigned next?
2. In what order should we try the values?
3. Can we detect inevitable failure early?
4. Can we take into account problem structure?
Constraint graph

- Nodes: variables
- Arcs: (binary) constraints
Most constrained variable:

- choose the variable with the fewest legal values, a.k.a. minimum remaining values (MRV) heuristic
**Most constraining variable**

- Tie-breaker among most constrained variables
- **Most constraining variable:**
  - choose the variable with the most constraints on remaining variables
Least constraining value

Given a variable, choose the least constraining value (the one that rules out the fewest values in the remaining variables)

Combining these heuristics makes 1000 queens feasible
Improving backtracking search

General purpose methods can drastically improve speed

- Which variable should be assigned next?
  - Most constrained → Most constraining

- In what order should we try the values?
  - Least constraining

- Can we detect inevitable failure early?

- Can we take into account problem structure?
Forward checking

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

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

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

**Idea:**
- Keep track of remaining legal values for unassigned variables
- Terminate search when any variable has no legal values
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!
- Can use constraint propagation to detect violations early
Arc consistency

- Simplest form of propagation makes each arc consistent
- $X \rightarrow Y$ is consistent iff for every value $x$ of $X$ there is some allowed $y$
Arc consistency

- Simplest form of propagation makes each arc consistent
- $X \rightarrow Y$ is consistent iff
  for every value $x$ of $X$ there is some allowed $y$

If $X$ loses a value, neighbors of $X$ need to be rechecked
Arc consistency

- Simplest form of propagation makes each arc **consistent**
- \( X \rightarrow Y \) is consistent iff
  
  for every value \( x \) of \( X \) there is some allowed \( y \)

If \( X \) loses a value, neighbors of \( X \) need to be rechecked
Arc consistency

- Simplest form of propagation makes each arc consistent
- \( X \rightarrow Y \) is consistent iff
  - for every value \( x \) of \( X \) 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
- Can be run as a preprocessor or after each assignment

Only need to recheck arc \( X_i \rightarrow X_j \) when \( X_i \) lost some values

\[ \Rightarrow \text{at most } d = \text{size of domain of } X_i \text{ times} \]
Arc consistency algorithm AC-3

```
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

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
```

Complexity = \(O(c \cdot d^3)\)  
\(c\): # constraints 
\(d\): max domain size
Improving backtracking search

General purpose methods can drastically improve speed

- Which variable should be assigned next?
  - Most constrained ➔ Most constraining

- In what order should we try the values?
  - Least constraining

- Can we detect inevitable failure early?
  - Forward checking, constraint propagation

- Can we take into account problem structure?
Problem structure

- Constraint graph

Suppose we have $n$ variables, grouped into independent subproblems with at most $c$ variables.

Size of search tree:

$$\frac{n}{c} \cdot d^c \ll d^n$$
Theorem: If CSP has tree structure, can solve it in time $O(n \cdot d^2)$

Will see this again for probabilistic reasoning!
Solving tree structured CSPs

- Choose root; orient edges away from root
- Pick topological ordering
- For j from n down to 1: remove all parent values for which there is no consistent child value
- For j from 1 to n: assign values consistently with parent

Special case of constraint propagation
Nearly tree-structured CSPs
Cutset conditioning

- Pick subset ("cutset") of variables such that remaining variables form a tree
- Search through each possible instantiation of cutset, and try to solve remaining tree-structured CSP

Complexity: Suppose we know the cutset already

\[ d^k \cdot (n-k) \cdot d^2 \]

\# of solving the subproblems tree problem on n-k
Junction trees (more later)
Improving backtracking search

General purpose methods can drastically improve speed

- Which variable should be assigned next?
  - Most constrained ➔ Most constraining

- In what order should we try the values?
  - Least constraining

- Can we detect inevitable failure early?
  - Forward checking, constraint propagation

- Can we take into account problem structure?
  - Independent subproblems; trees; tree-like graphs
CSPs are special search problem
- Environment state described using variables
- Goal test given by constraints

Backtracking = DFS with fixed var. assigned per node

Can be sped up using
- Variable and value selection heuristics
- Forward checking
- Constraint propagation / inference
- Exploit dependency structure among variables