Introduction to Artificial Intelligence

Lecture 5 – CSP

CS/CNS/EE 154
Andreas Krause
Announcements

- Sign up for projects
  - Will make assignments tomorrow
- Homework 1 is out. Due Friday Oct 22
Games vs. search

- In games, actions are **nondeterministic**
  - Opponent can affect state of the environment
- Optimal solution no longer sequence of actions, instead a **strategy** (policy, conditional plan)
  - If you X I’ll do Y, else if you do Y I’ll do Z, ....
Minimax game tree

- Search for optimal move no matter what opponent does
- minimax value = best achievable payoff against best play

Can evaluate using DFS
α-β-pruning

Key idea: For each node n in minimax tree keep track of

- α: Best value for MAX player if n is reached
- β: Best value for MIN player if n is reached

Never need to explore consequences of actions for which β<α

Avoid exploring “provably suboptimal” parts of minimax tree
Nondeterministic and partially observable search

- Nondeterminism
  - Environment state not a function of current state and action
- Partial observability
  - Percept is not a function of environment state

![Diagram showing agent and environment interactions](attachment:image.png)
Nondeterminism

So far, assumed that
- all actions are deterministic
- State is fully observable

What if the actions are noisy?

Example:
- If applied on dirty square: “Suck” sometimes cleans up neighboring square as well
- If applied on clean square: “Suck” sometimes dirties the square

Solution?
AND-OR trees
Conditional plans

- For nondeterministic actions, optimal solution no longer a sequence, but a conditional plan (strategy)
- Can represent in AND-OR tree (like minimax)
- **OR nodes**: Agent chooses action
- **AND nodes**: Environment chooses next state
  - Need to specify what to do for every possible next state!
- Evaluate using backward induction (like minimax)
- Use IDS to grow tree until found solution (or tired)
Partial observability

Suppose our robot is sensorless. Can we still plan?

Plan on “belief states” (sets of environment states), instead of individual states
Working with belief states
Noisy actions (slippery vacuum world)
Planning in belief space
Planning in belief space

- Belief state = set of states agent could be in
- Belief state is a goal if all contained states are goals
- Successor function keeps track of all states the agent could be in after taking a particular action ("prediction")
Sensing: Incorporating observations

Even with deterministic actions, percepts can be nondeterministic → Need conditional plan
Sensing: Incorporating observations

Right

[B, Dirty]

[A, Dirty]

[B, Clean]
AND-OR trees with noisy observations

Diagram showing a tree structure with nodes labeled 1, 2, 3, 4, 5, and 7, and actions labeled 'Suck' and 'Right'. Nodes are connected with arrows indicating the flow of actions.
Can reduce any partially observable problem to fully observable problem on belief states

Belief state = set of states the environment could be in

Use existing algorithms on belief states
  - Sensor-less case: Find opt. sequence using IDS, A*, etc.
  - Observations: Develop conditional plans using AND-OR search
  - Games: Use α-β pruning etc. on belief states

# belief states exponential in # states...

Need concise way to summarize the states we could be in ➔ logic! (coming up soon)
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, Y \}

Constraints:
WA ≠ NT ∧ WA ≠ SA ∧ ...
(\{ WA, NT \} ∨ \{ R, Y \}, \{ R, B \}, \{ Y, B \}, \{ Y, R \}, \{ B, R \}, \{ B, Y \})

• Variables? Domains? Constraints?
Example: Sudoku

Variables: $X_{ij}$; $i,j \in \{1, \ldots, 9\}$

Domain: $\{1, \ldots, 9\}$

Constraints: $\text{AllDiff}(X_{11}, X_{22}, \ldots, X_{99}) \land \text{AllDiff} \ldots$

(never have number occ. 2x per col-/row/block)
Example: 8 queens

Variables: $X_i$: pos. of queen in col. $i$

Domains: $\{1, \ldots, 8\}$

Constraint: No queens attack each other.
Example: k-SAT

Disjunction of 3 literals

\[(X_1 \lor \neg X_3 \lor X_7) \land (\neg X_1 \lor X_4 \lor X_8) \land \cdots \land (X_5 \lor \neg X_{17} \lor \neg X_{21})\]

Vars: \(X_1, \ldots, X_n\), Domains: \{true, false\}, Constraints

- Fundamental special case
- All variables binary
- Constraints: disjunctions of \(k\) (negated) variables

- NP-complete for \(k>2\)
- Polynomial time solvable for \(k=2\)
Types of CSPs

- **Discrete variables**
  - Finite domains: Map coloring, Sudoku, 8 queens, SAT
  - Infinite domains: $X_2 \geq X_1 + 3 \land X_5 \geq X_3 + 2$

- **Continuous variables**
  - Job scheduling
  - $K_i$: start time of job $i$, Domain: [1, 2, ..., 3]

- Continuous variables
  - Robot (factory control, time fading, ...
Types of constraints

- **Unary**: involve single variable  \( E.g.: \ NSW = B \)

- **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
Example: higher-order constraints

\[
\begin{array}{c}
T \quad W \\
+ T \quad W \\
\hline
O \quad O \\
\end{array}
\]

\[
F \quad T \quad U \quad W \quad R \quad O
\]

\[O + O = R + 10 \cdot X_1\]
\[w + w + x_1 = U + 10 \cdot X_2\]
\[\vdots\]
\[\text{All Diff} (O, U, R, W, T, F)\]
Example: higher-order constraints
Real-world examples of CSPs

- Assignment problems
- Timetabling
- Hardware configuration
- Transportation scheduling
- Multi-agent coordination
- ...

29
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?

Size of search tree: $O(n! \cdot d^n)$
Backtracking search

- Variable assignments are commutative!

\[ \lfloor \text{NSW = Y then NT = B} \rfloor \text{ same as } \lfloor \text{NT = B then NSW = Y} \rfloor \]

- 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?