Created by Bay Wei Heng

Winograd Schema - A sentence with an ambiguous pronoun, need to decide what the pronoun refers to. This depends on context which can easily be flipped by a single word.

An agent is an entity that perceives and acts. Abstractly, it is a function from percept histories to actions. Agent = architecture + program.

e.g. Percepts: location and cleanliness, Actions: Left, Right, Suck, NoOp

Performance measure: objective criterion for measuring success of an agent's behavior.

Rational agent aims to maximize performance measure.

An agent is autonomous if its behavior is determined by its own experience (with ability to learn and adapt)

Specifying task environment: PEAS - Performance measure, Environment, Actuators, Sensors

Properties of task environments:

- Fully observable (vs. partially observable): Sensors provide access to the complete state of the environment at each point in time
- Deterministic (vs. stochastic): The next state of the environment is completely determined by the current state and the action executed by the agent
- Episodic (vs. sequential): Current action does not affect future decisions
- Static (vs. dynamic): The environment is unchanged while an agent is deliberating
- Discrete (vs. continuous): A finite number of distinct states, percepts and actions
- Single-agent (vs. multi-agent): An agent operating by itself in an environment

Four basic types of agents: simple reflex, model-based reflex, goal-based, and utility-based

In a fully observable, deterministic, discrete environment, we can "search for solutions". Problem formulation: Goal states (either explicit e.g. Work or implicit e.g. function isCheckmate). Path cost (additive, assumed to be ≥ 0).

Tree search: Start at source, keep searching until reach goal. Frontier is defined as the nodes we have seen but have not explored yet (at initialization, frontier = {source}). At each iteration, choose a node from frontier, explore it, and add its neighbors to the frontier.

Graph search: A node that's been explored once will not be revisited

A state represents a physical configuration (e.g. 8-puzzle layout). A node is a data structure constituting part of search tree. It includes state, parent node, action, and path-cost. Two different nodes can contain the same world state.

Evaluation Criteria:

- completeness: always find a solution if it exists
- optimality: find a least cost solution
- time complexity: number of nodes generated
- space complexity: max number of nodes in memory

Problem Parameters: b: maximum number of successors of any node d: depth of shallowest goal node m: maximum depth of search tree (may be ∞)

Uninformed search only uses problem input

- Breadth-first search - Complete if b finite, not optimal unless step cost is constant, $O(b^d)$ time and space
- Uniform-cost search (Dijkstra) - Complete (if all step costs $\geq \epsilon$), optimal, $O(b^{1+C^*/\epsilon})$ time and space, where C^* is optimal cost
- Depth-first search - Not complete on infinite depth graphs, non-optimal, time $O(b^m)$, space $O(bm)$ but can be $O(m)$
- Depth-limited search - Same as DFS, replace m with l. Might not find goal if $l \leq d$.
- Iterative deepening search - Same as BFS, except $O(bd)$ or $O(d)$ space.

h is admissible if for all n, $h(n) \leq h^*(n)$ where $h^*(n)$ is the true cost to reach the goal state from n i.e. a heuristic is admissible if it never overestimates the cost to the goal (e.g. straight line).

If h admissible then A^* using TREE-SEARCH is optimal.

h is consistent if, for every node n and every successor n' of n generated by any action a, we have $h(n) \leq d(n, n') + h(n')$ i.e. the heuristic function decreases by at most the edge weight whenever we traverse an edge.

If h is consistent then h is admissible, and A^* using GRAPH-SEARCH is optimal.

If $h_2 \geq h_1$ then h_2 dominates h_1. h_2 will incur lower search cost than h_1.

Informed search

- Best-first search - use an evaluation function f for each node n. Expand argmin$f(n)$ first. Special cases are:
- Greedy best-first search - expands the node that appears to be closest to goal, i.e. take $f(n) = h(n)$, where h is heuristic function. Same properties as BFS.
- A^* search avoids searching already expensive paths, sets $f = g + h$ where $g(n)$ is cost from reaching n from start node. It is complete if there is a finite number of nodes with $f(n) \leq \inf_g f(g)$, takes $O(b^{h^*(s_0)-h(s_0)})$ time and $O(b^m)$ space.

To derive an admissible heuristic, relax the problem by removing restrictions on actions. The cost of an optimal solution to the relaxed problem is an admissible heuristic to the original problem.

Local search algorithms maintain single "current best" state and try to improve it - finds reasonable solutions in large state space using very little/constant memory. Used when the path to goal is irrelevant i.e. the goal state itself is the solution.

E.g. Hill-climbing search - Keep moving to the highest-valued successor until a local maximum is reached. To avoid getting stuck at local maxima, can try random restarts and sideway moves. For 8-Queens we can reach a local minimum of one conflicting pair, then not be able to move any queen.

Adversarial search

Games - have a utility maximizing opponent, solution is a strategy specifying a move for every possible opponent response. time limit - unlikely to find goal and must approximate.

Game is defined by 7 components

- Initial state s_0
- States S
- Players: $PLAYER(s)$ defines which player has the move in state s
- Actions: $ACTIONS(s)$ returns set of legal moves
- Transition model: $RESULT(s, a) \in S$
- Terminal test: $IS_TERMINAL(s)$
- Utility function: $UTILITY(s, p)$

A strategy for player i specifies what action he will take in every node that he makes a move in.

A strat s_1^* is winning if for any strategy s_2 by player 2, the game ends with player 1 as winner. Similar definition for non-losing.

Von Neumann Theorem: Exactly one of the following is true in chess: White has a winning strategy, Black has a winning strategy or each player has a non-losing strategy.

Minimax play achieves a subgame perfect Nash equilibrium, prove by backward induction. It is complete if game tree is finite, optimal, $O(b^m)$ time and $O(bm)$ space.

α-β pruning - maintain lower bound α and upper bound β of the values of MAX and MIN's (resp.) nodes seen so far, prune subtrees not affecting minimax decision. Perfect ordering time complexity is $O(b^{m/2}) = O((\sqrt{b})^m)$. Random ordering $O(b^{3m/4})$.

```
int max_value(s,a,b) {
   if(terminal(s)) return utility(s);
   int v = NEG_INFTY;
   for(nx: neighbours(s)) {
      v = max(v, min_value(nx,a,b));
      if(v >= b) return v;
      a = max(a, v);
   }
   return v;
}
```

Even with α-β pruning, depth of tree might still be huge. Hence can use evaluation function and cutoff test e.g. depth limit. Instead of returning utility at terminal, return evaluation at cutoff instead. Evaluation function maps game states to reals. Ideally strongly correlated with chances of winning at non-terminal states. Evaluation function need not return actual expected values, just maintain relative order of states. Cutoff depth limit can be combined with iterative deepening.

For stochastic games we add chance layers, with probabilities of going to different states.

Constraint Satisfaction Problems are specified by

- Variables $\overrightarrow{X} = X_1, X_2, \ldots, X_n$, each X_i has a domain D_i.
- Constraints $\overrightarrow{C}$: what variable combinations are allowed? Constraint scope - what variables are involved. Constraint relation - what is the relation between them.
- Objective is to find a legal assignment to values to variables i.e. $y_i \in D_i$ with all constraints satisfied.

Linear Programming - variables are all rational values, linear constraints Combinatorial optimization - Graph problems like vertex cover, bipartite matching, minimum spanning tree, operations research problems like scheduling, bin packing, task allocation

E.g. For graph coloring, variables are vertices, domains are set of colors, constraints - if $(u, v) \in E$ then $c(u) \neq c(v)$

For Binary CSP each constraint relates two variables - can form constraint graph. In general can form hypergraph. Unary constraints like $c(v) \neq G$ also possible.

Can have disjunctive constraint.

CSP can have discrete variables from finite or infinite domains, or continuous variables.

Standard search formulation (incremental): states are partial assignments, initial state empty, every solution occurs at depth n (when all vars assigned). Order of variable assignment is irrelevant, so can backtrack when no legal assignments.

Heuristics: Which variable should be selected next? In what order should values be tried? How can domain restrictions on unassigned variables be inferred? Can we detect inevitable failure early?

- Most constrained variable: choose var with fewest legal values
- Least constraining variable: choose var that rules out fewest values for neighbors
- Forward checking: Keep track of remaining legal values for unassigned variables, terminate search early when any variable has no legal values.
- Constraint propagation: Like FC, but repeatedly locally enforces constraints.
- Arc-consistency: The arc (X_i, X_j) is consistent iff for every value $x \in D_i$ there exists $y \in D_j$ that satisfies the arc constraint. Simplest form of CP makes each arc consistent - make arcs directed (binary constraint becomes two arcs). Shrink domains repeatedly until so. If D_i loses a value, X_i's neighbors need to be rechecked. AC propagation detects failure earlier than forward checking. Can be run as a preprocessing step or after each assignment. Time complexity is $O(n^2d^3)$, since CSP has at most n^2 directed arcs, each arc can be inserted at most d times since D_j has at most d values to delete, and checking the consistency of each arc takes $O(d^2)$ time.
- Maintaining AC - AC but after each choice of variable we maintain AC again.
- Special types of consistency - AllDiff constraint can be reduced to $nC2$ binary constraints, but it is better to use a special-purpose algorithm instead. Similar for k-consistency
- Local search for CSP: allow states that violate constraints, actions can reassign to variables. e.g. select random conflicted variable and reassign it randomly e.g. hill-climb with $h(n) = \#$violated constraints

Theorem: If CSP graph is a tree, we can compute a satisfying assignment (or decide none exist) in $O(nd^2)$ time. 1. Make parents arc-consistent with children (nd^2) 2. Assign values from root down (nd)

Nash Equilibrium: Players knows the strategies of all opponents: no one wants to change her strategy (no higher utility). Subgame perfect NE, every subgame is a NE.