Agents

Rational: Maximally achieving goals (actions that maximize utility function)

Reflex Based: Chooses action based on current percept (no future consideration)

Goal Based: Chooses action based on consequences (model of how the world reacts)

Utility Based: Goal based with trading off of multiple goals and uses probabilities

Search

Def: Possible states, Successor function $f(n) \rightarrow (n'$, action, cost), start and goal state

Complete: Guaranteed to find a solution if one exists

Optimal: Guaranteed to find the least cost path

Properties: n= number of states, b= maximum branching factor, C^*= optimal cost, d= depth of shallowest solution, m= max depth, ϵ = min cost of all actions

Conformant Planning: Set of actions that always work (sterilizing surgical gear)

Blind Search

DFS: Fringe uses a Stack, complete iff finite, not optimal, time: O(b^m), space: O(bm)

BFS: Fringe uses a Queue, complete, optimal (constant), time and space: O(b^d)

IDDFS: Fringe uses a Stack, complete, optimal (constant), time: O(b^d), space: O(bm)

Heuristic Search

Heuristic $h(n)$ = An estimate of how close a state is to a goal

Admissible: Always an underestimate to the true lowest cost

Consistent: Always $h(n) <= h(n') + stepCost(n')$ where n' is a neighbor of n

Best First: Fringe uses a PriorityQueue with cost fuction $f(n)$ for each node

Uniform Cost: Best First with $f(n)$ = sum of edge costs from start to n (explores increasing contours), complete, optimal, time and space: O($b^{\frac{C^*}{\epsilon}}$)

Greedy: Best First with $f(n) = h(n)$ (suboptimal goal is common)

A*: Best First with $f(n) = g(n) + h(n)$ with $g(n)$ = sum of costs from start to n

IDA*: Depth bound is now $F_{limit} = h(start)$, prune if $f(n) > F_{limit}$, $F'_{limit} = min(pruned\ nodes)$, uses space of DFS, time depends on # of unique F values

Beam: Best First with $|Fringe| = K$, not complete, time: O(b^d), space: O($b + K$)

Hill Climbing: Always choose best child (Beam Search with K = 1)

Tabu: Keep fixed length queue of states to not visit again (use with hill climbing)

Stochastic Search

Hill Climbing++ Restarts: Generate random state when plateaued

Hill Climbing++ Walk: With prob p move to the neighbor with largest value, with $(1 - p)$ move to a random neighbor

Hill Climbing++ (Both): Greedy move, random walk, or random restart

Simulated Annealing: Pick a random neighbor and calculate the change in 'energy' or objective function δ, if it is positive then move to that state. Otherwise, move to this state with probability $e^{\frac{\delta}{T}}$ where T is decreased as the algorithm runs longer. High T $\rightarrow$ probability of bad move is higher and vice versa

Genetic: Start with a population of random states, use an evaluation (fitness) function, produce next generation using random selection / crossover / random mutation

Gradient Descent: Move in the direction of the gradient at each step

Constraint Satisfaction Problems (CSPs)

Def: Goal test is a set of constraints over the state's variables $x_i \in D_i$ *or* D

Constraint Graphs: Nodes are variables, (multi)edges show constraints

As a Search Problem: States defined by the values assigned so far, initially empty, Successor function assigns a value to an unassigned variable, and the Goal test checks to see if the current assignment is complete and satisfactory

Improvements: Fix ordering with variable assignments, check constraints as you go

Forward Checking: Cross off values that violate a constraint when added to the existing assignment (Immediate neighbors and fail if the set of possible values is empty)

Arc Consistency: 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

Constraint Propagation: If X loses a value, neighbors of X need to be rechecked

Min Remaining Values: Choose the variable with fewest legal values in its domain

Max Degree: Choose the variable in the most constraints with remaining variables

Least Constraining Value: Given a variable, assign a value that rules out the fewest values in remaining variables

Rationale: Want to enter most promising branch, but detect failure quickly. MRV and MD choose the variable most likely to cause failure while LCV ensures that an early value choice does not cause failure later

Constraint Satisfaction Problems II (CSPs)

Trees: Choose a root variable, order the other variables so parents precede children. Remove Backward: For i = n : 2, apply RemoveIncosistent(Parent(x_i), x_i) then Assign Forward: For i = 1 : n, assign x_i consistently with Parent(x_i). Time: O(nd^2)

Proof: After backward pass, all root-to-leaf arcs are consistent, no backtracking

Nearly Tree-Structured: instantiate (in all ways) a set of variables such that the remaining constraint graph is a tree, Time: O($d^c(n-c)d^2$)

Iterative Improvement: Randomly select conflicted variable, choose a value that violates the fewest constraints (hill climb with $h(n)$ = number of violated constraints)

Adversarial Search

Minimax: The opitmal utility with a rational adversary, Time: O(b^m), Space: O(bm)

Pruning: α is Max's best choise on a path to root. If the best value becomes worse than α, no point in exploring children. Similiar for β. Time: O($b^{m/2}$)

```
def value(state, agent, alpha, beta):
    for each successor of state:
        if agent == 0:
            v = max(v, value(successor, agent + 1, alpha, beta))
            if v >= beta: return v
            alpha = max(alpha, v)
        else:
            v = min(v, value(successor, agent + 1, alpha, beta))
            if v <= alpha: return v
            beta = min(beta, v)
    return v
```

Heuristic Evaluation: Scores non-terminals and returns utility of the state

Utility: Function from state to real numbers that describe an agent's preferences

Expectimax: Instead of the min value being chosen, the expected value for that state is returned: $\sum_i [value(successor_i) * P(successor_i)]$

Preferences: A is preferred to B: $A \succ B$, A is indifferent to B: $A \sim B$

Rationality: Orderability: $(A \succ B) \vee (B \succ A) \vee (A \sim B)$,

Transitivity: $(A \succ B) \wedge (B \succ C) \implies (A \succ C)$,

Continuity: $A \succ B \succ C \implies \exists_p [p, A : 1-p, C] \succ B$,

Substitutability: $A \sim B \implies [p, A : 1-p, C] \sim [q, B : 1-q, C]$,

Monotonicity: $A \succ B \implies (p \geq q \iff [p, A : 1-p, B] \succeq [q, A : 1-q, B]$

Markov Decision Processes

Def: states, actions, Transition $T(s,a,s')$ or $P(s'|s,a)$, Reward $R(s,a,s')$ or $R(s)$

Policy: $\pi^* : S \to A$, gives an optimal action for all states

Discounting: Each time a level is descended multiply by another factor of γ, Sooner rewards have higher utility than later rewards and helps algorithm converge: $U([r_0, r_1, ...]) = r_0 + \gamma r_1 + \gamma^2 r_2 + ...$

Bellman Equations: $V^*(s) = \max_a Q^*(s,a)$ = utility starting in s and then acting optimally, $Q^*(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$ = utility starting after taking action a in s then acting optimally

Value Iteration: $V_0(s) = 0$, given $V_k(s)$ simultaneously solve one ply of expectimax from each state: $V_{k+1}(s) \leftarrow \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_k^*(s')]$ with Time: $O(|S|^2|A|)$

Policy Extraction: Given the optimal values, $V^*(s)$, $\pi^*(s) = \operatorname{argmax}_a Q^*(s,a)$

Problems: Slow, max at each state rarely changes and the policy often converges long before the values do

Asynchronous Value Iteration: Not essential to back up all states in each iteration as long as no state is starved. Backup with a heap of states ordered by the expected change in value (prefer backing a state whose successors had most change

Policy Evaluation: compute the utility of a state s under a fixed policy: $V^\pi(s)$ = expected total discounted rewards starting in s and following π

Policy Iteration: Initialize $\pi(s)$ to random actions, calculate utilities of π at each s using a nested loop, update policy using one-step look-ahead to see what's the best action I could execute, assuming I then follow $\pi(s)$

Initialize $\pi_i(s)$ to random actions and $i = 0$. Repeat:

Step 1: Policy Evaluation:

Initialize $k = 0$ and for each s, $V_0^\pi(s) = 0$ and repeat until V^π converges:

For each s $V_{k+1}^{\pi_i}(s) \leftarrow \sum_{s'} T(s, \pi_i(s), s')[R(s, \pi_i(s), s') + \gamma V_k^{\pi_i}(s')]$

$k++$

Step 2: Policy Improvement:

For each s, $\pi_{i+1}(s) = \operatorname{argmax}_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^{\pi_i}(s')]$

If $\pi_i == \pi_{i+1}$, then it is optimal

Else let $i++$