SEARCHING

Uninformed Search

Depth-First Search: Strategy: expand a deepest node first. Implemented as LIFO stack. Very space (memory) efficient.

Breadth First Search: Strategy: expand the shallowest node first. Implemented as FIFO queue. Finds the shortest path in terms of number of actions.

Iterative Deepening: Combines the advantage of DFS's space and BFS's time/shallow-solution.

Uniform Cost Search (UCS): Like BFS, but takes cost (edge weights) into account. Expand a cheapest (lowest cost) node first. Fringe is a priority queue. UCS is optimal and complete.

Informed Search

Informed search algorithms involve the concepts of "direction" which can be fed to the algorithm through heuristics. A heuristic is a function that estimates how close a state is to a goal, and is designed for a particular problem.

Greedy Search (Best-First): Expands the node that seems the closest to a goal state. It can provide wrong solutions.

A* Search: Combines uniform cost search (backward cost *g(n)*) and greedy search (forward cost *h(n)*). A* expands toward the goal and hedge its bets to ensure optimality as opposed to uniform cost which expands equally in all directions. To have an optimal solution in tree search, *h(n)* must be admissible. *0 <= h(n) <= h*(n)* where *h*(n)* is the true cost to a nearest goal.

In graph search, the heuristics must be consistent which implies admissibility.

Adversarial Search (MinMax)

Players alternate turns and the agent computes each node's minmax value (the best achievable utility). Example: tic-tac-toe, chess. **Alpha Beta Pruning:** Remove nodes that will not affect the final outcome.

Local Search

Hill Climbing: 1. Start from any state 2. Repeat until no neighbors better than current: move to best neighboring state.

Simulated Annealing: Escapes local maxima by allowing downhill moves.

Constraint Satisfaction Problem

A state is defined by variables X_i with values from a domain *D*. Goal test is a set of constraints specifying allowable combinations of values for subsets of variables. It can be implemented using graph structure to speed up the search. Variables can discrete (finite or infinite domains) or continuous. Constraints can be unary, binary, higher order constraints. Constraints can be soft. Example: map coloring, n-queen, Sudoku, assignment problem.

Backtracking uninformed search can be used to solve CSPs. It is the same as depth first search with two improvements: 1. Get one variable at a time 2. Check constraints as you go. Backtracking can be further improved using ordering (minimum remaining values/least constraining value) and filtering (forward checking/constraint propagation) concepts (removing domain values that already do not match).

MAKING DECISIONS

Markov Decision Processes

An MDP is defined by a set of states, set of actions, transition function, reward function, and start state. It is non-deterministic search problem. It is used to compute optimal values using value iteration or policy iteration. It can be used compute values for a particular

policy using policy evaluation or turn values into policy using policy extraction. **Discounting:** Quantifies the preference for immediate rewards rather than long-term rewards.

Recursive definitions of values:

$$V^*(s) = \max_a Q^*(s,a)$$

$$Q^*(s,a) = \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s')\right]$$

$$V^*(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V^*(s')\right]$$

Value iteration: Start with $V^0(s) = 0$: no time steps left means Given vector of $V^k(s)$ values, do one ply of expectimax from each state: an expected reward sum of zero. Repeat until convergence.

Policy Evaluation: Evaluate one given policy. This is easier: we know the action, so we do not need **max** function. Solve it as a set of linear equations!

Policy Iteration: Start with a random policy. In a loop, do:

(i) Evaluate current policy
(ii) Extract the new policy based on values
(iii) Change that to the new policy

The new policy will be better (or we're done).

LEARNING

Reinforcement Learning

We assume a Markov decision process and are looking for optimal policy. But we don't know the model T or the reward R. **Model-Based Learning:** 1. Learn an approximate model based on experiences 2. Solve for values as if the learned model were correct. Direct evaluation: compute values (average observed sample values) for each state under a policy.

Model-Free Learning: Learn the Q/V values directly. Example: **Q-Learning:** Sample-based Q-value iteration.

1. Receive a sample (s,a,s',r)
2. Consider your old estimate. *Q(s,a)*
3. Consider your new sample estimate. *sample = R(s,a,s') + γ max Q(s',a')*
4. Incorporate the new estimate into a running average.
   Q(s,a) ← *(1-α) Q(s,a) + α [sample]*

Exploration vs. Exploitation: Quantitatively balance exploration (trying new actions) and exploitation (making the best-known action). We can use ε-greedy algorithm, use exploration functions, minimize Regret, etc.

Download again? https://is.gd/q4uANh

Markov Models

We want to reason about a sequence of observations. Same as MDP transition model, but no choice of action. Joint distribution can be written as:

$$P(X_1, X_2, \ldots, X_T) = P(X_1)P(X_2|X_1)P(X_3|X_2)\ldots P(X_T|X_{T-1})$$
$$= P(X_1)\prod_{t=2}^{T} P(X_t|X_{t-1})$$

Bayes Theorem

How to write p(A/B) in terms of p(B/A). Generally, we know the probability of effect given a cause. We usually want to know the probability of a cause, when observing an effect.

Hidden Markov Models

Underlying Markov chain over states X. You observe outputs (effects or emissions) at each time step. Example: weather

We use observations to update belief.

Joint distribution:

$$P(X_1, E_1, \ldots, X_T, E_T) = P(X_1)P(E_1|X_1)\prod_{t=2}^{T} P(X_t|X_{t-1})P(E_t|X_t)$$

We use the Viterbi Dynamic Programming Algorithm to find the Most Likely Explanation (MLE).