Columbia CS188 - Artificial Intelligence

Marco Filippone

December 9, 2020

1 Introduction

Artificial intelligence is the study and design of intelligence agents where an intelligent agent is a system that perceives it's environment and takes actions that maximizes its chances of success.

2 Rational Agents

Rational Agent For each possible percept sequence, a rational agent should select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has.

When we define a rational agent, we group these properties under PEAS:

- Performance
- Environment
- Actuators
- Sensors

the problem specification for the task environment. The rational agent we want to design for this task environment is the solution.

2.1 Environment

- Fully observable (vs. partially observable): An agents sensors give it 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. (If the environment is deterministic except for the actions of other agents, then the environment is strategic)
- Episodic (vs. sequential): The agents experience is divided into atomic episodes (each episode consists of the agent perceiving and then performing a single action), and the choice of action in each episode depends only on the episode itself.
- Static (vs. dynamic): The environment is unchanged while an agent is deliberating. (The environment is semi-dynamic if the environment itself does not change with the passage of time but the agents performance score does.)
- Discrete (vs. continuous): A limited number of distinct, clearly defined percepts and actions. E.g., checkers is an example of a discrete environment, while self-driving car evolves in a continuous one.
- Single agent (vs. multi-agent): An agent operating by itself in an environment.
- Known (vs. Unknown): The designer of the agent may or may not have knowledge about the environment makeup. If the environment is unknown the agent will need to know how it works in order to decide.

2.2 Agents

The following agents types are listed in order of generality.

2.2.1 Simple reflex agents

- select an action based on the current state only ignoring the percept history
- Can only work if the environment is fully observable

2.2.2 Model-based reflex agents

- Handle partial observability by keeping track of the part of the world it cant see now.
- Internal state depending on the percept history (best guess).
- Model of the world based on how the world evolves independently from the agent, and how the agent actions affects the world

2.2.3 Goal-based agents

- Knowing the current state of the environment is not enough. The agent needs some goal information.
- Agent program combines the goal information with the environment model to choose the actions that achieve that goal.
- Flexible as knowledge supporting the decisions is explicitly represented and can be modified.

2.2.4 Utility-based agents

- Sometimes achieving the desired goal is not enough. We may look for quicker, safer, cheaper trip to reach a destination.
- Agent happiness should be taken into consideration. We call it *utility.*
- A utility function is the agents performance measure
- Because of the uncertainty in the world, a utility agent choses the action that maximizes the expected utility.

2.2.5 Learning agents

Four conceptual components:

- Learning element: responsible for making improvements
- Performance element: responsible for selecting external actions. It is what we considered as agent so far.

- Critic: How well is the agent is doing w.r.t. a fixed performance standard.
- Problem generator: allows the agent to explore.

2.3 Agent's organization

- Atomic Representation: Each state of the world is a blackbox that has no internal structure.
- Factored Representation: Each state has some attributevalue properties
- Structured Representation: Relationships between the objects of a state can be explicitly expressed.

3 Search Agents

Applications

- Route-finding problem
- Travelling salesmane person
- VSLI layout: position million of components and connections on a chip to minimize area, shorten delays. Aim: put circuit components on a chip so as they dont overlap and leave space to wiring which is a complex problem.
- Robot navigation
- Automatic assembly sequencing
- Protein folding

3.1 Problem formulation

- Initial state: the state in which the agent starts
- States: All states reachable from the initial state by any sequence of actions (State space)
- Actions: possible actions available to the agent. At a state s, $Actions(s)$ returns the set of actions that can be executed in state s. (*Action space*)
- Transition model: A description of what each action does $Results(s, a)$
- Goal test: determines if a given state is a goal state
- Path cost: function that assigns a numeric cost to a path w.r.t. performance measure

3.2 Search Space

- State space: a physical configuration
- Search space: an abstract configuration represented by a search tree or graph of possible solutions.
- Search tree: models the sequence of actions
  - Root: initial state
  - Branches: actions
  - Nodes: results from actions. A node has: parent, children, depth, path cost, associated state in the state space.
- Expand: A function that given a node, creates all children nodes

The search space is divided into three regions:

- Explored (a.k.a. Closed List, Visited Set)
- Frontier (a.k.a. Open List, the Fringe)
- Unexplored.

3.3 Search Strategies

Strategy evaluation Strategies are evaluated along the following dimensions:

- Completeness: Does it always find a solution if one exists?
- Time complexity: Number of nodes generated/expanded
- Space complexity: Maximum number of nodes in memory
- Optimality: Does it always find a least-cost solution?

Time and space complexity: are measured in terms of:

- b: maximum branching factor of the search tree (actions per state).
- d: depth of the solution
- m, or D: maximum depth of the state space (may be ∞).

4 Uninformed search

Let:

- b is the average branching factor
- d is the depth level
- m is the maximum depth of the search space

Then:

Strategy	Complete	Time	Space	Optimal	Implementation	Comment
BFS	If b is finite	$O(b^d)$	$O(b^d)$	Yes, if uniform cost	FIFO	High memory, exponential time
DFS	No, fails in: infinite depth spaces and spaces with loops	$O(b^m)$	$O(bm)$	No	LIFO	Bad if m is much larger than d but if solutions are dense, may be much faster than BFS
Depth limited Search					DFS with limit	
Iterative Deepening					Depth deepening with increasing limits	Benefits of BFS and DFS
Uniform cost	Yes, if solution has finite cost	$O(b^{C^*/\varepsilon})$	# of nodes with $g \leq C^* \to O(b^{C^*/\varepsilon})$	Yes	BFS expanding by cost of path (not of last step)	

Table 1: Uninformed Search Strategies

5 Informed Search

5.1 Heuristics

Admissible heuristic An admissible heuristic never overestimates the cost to reach the goal, that is it is optimistic. A heuristic h is admissible if: $\forall\, node\, n,\ h(n) \leq h^*(n)$ where h^* is true cost to reach the goal from n.

In a map, the air distance between two points is an admissible heuristic because it is by definition the shortest distance between the two points.

Strategy	Implementation
Greedy search	Expands the node that appears to be closest to goal
A^*	Minimize the total estimated solution cost (f), that is the cost to reach node n (g), and the cost to get from n to the goal (h)

Table 2: Informed Search Strategies

5.2 A*

- Complete
- Time: exponential
- Space: keeps every node in memory
- Optimal: If the heuristic function used by A* is admissible, then A* is admissible. An intuitive proof of this is as follows: When A* terminates its search, it has found a path from start to goal whose actual cost is lower than the estimated cost of any path from start to goal through any open node (the node's f value). When the heuristic is admissible, those estimates are optimistic (not quitesee the next paragraph), so $A*$ can safely ignore those nodes because they cannot possibly lead to a cheaper solution than the one it already has. In other words, A* will never overlook the possibility of a lower-cost path from start to goal and so it will continue to search until no such possibilities exist. (The actual proof is a bit more involved because the f values of open nodes are not guaranteed to be optimistic even if the heuristic is admissible. This is because the g values of open nodes are not guaranteed to be optimal, so the sum $g + h$ is not guaranteed to be optimistic)

6 Local Search

Idea: keep a single current state, and try to improve it. Move only to neighbors of that node. Advantages:

- No need to maintain a search tree.
- Use very little memory.
- Can often find good enough solutions in continuous or large state spaces.

6.1 Hill climbing (greedy local search)

- Looks only to immediate good neighbors and not beyond.
- Search moves uphill: moves in the direction of increasing elevation/value to find the top of the mountain.
- Terminates when it reaches a pick.
- Can terminate with a local maximum, global maximum or can get stuck and no progress is possible.
- A node is a state and a value.

Variants

- Sideways moves: escape from plateaux where best successor has same value as the current state
- Random-restart: overcomes local maxima, either find a goal or get several possible solution and pick the max
- Stochastic hill climbing chooses at random among the uphill moves.

6.2 Local Beam Search

- Maintains k states instead of one state.
- Select the k best successor, and useful information is passed among the states.

Stochastic beam search

- Choose k successors are random.
- Helps alleviate the problem of the the states agglomerating around the same part of the state space

6.3 Genetic algorithms

- A variant of stochastic beam search.
- Successor states are generated by combining two parents rather by modifying a single state.
- Starts with k randomly generated states, called **population**. Each state is an **individual**.
- The objective function is called **fitness function**
- Better states have high values of fitness function

Implementation

- Pairs of individuals are selected at random for reproduction w.r.t. some probabilities.
- A **crossover point** is chosen randomly in the string.
- Offspring are created by crossing the parents at the crossover point.
- Each element in the string is also subject to some **mutation** with a small probability.