Chapter 2: Intelligent Agents

(2.1): Agents and Environments Percept – content/information that agent's sensors are perceiving / capturing currently. **Percept Sequence** – a complete history of everything that agent has ever perceived. Action <-> percept sequence mapping IS the agent function. Agent function describes agent behavior. Agent function is an abstract concept. Agent program implements agent function.

(2.2): The Concept of Rationality A rational agent is one that acts to achieve the best outcome, or when there is uncertainty, the best expected outcome. Rationality maximizes what is EXPECTED to happen. Perfection maximizes what WILL happen.

(2.3): The Nature of Environments PEAS – Performance measure; Environment in which the agent will operate; Actuators that the agent will use to affect the environment; Sensors.

Properties

Fully vs partially observable. Single agent vs multiagent (comp vs. co-op). Deterministic vs. nondeterministic (stochastic/random). Episode vs. sequential. Static vs. dynamic. Discrete vs. continuous. Known vs. unknown (to the agent).

Deterministic means that the next state is completely determined by the current state and agent action whereas nondeterministic is NOT completely determined by the current state and agent action. Deterministic AND fully observable environment means there's no uncertainty, deterministic AND partially observable *may* appear nondeterministic.

Episodic means that agent experience is divided into individual, independent, and atomic episodes (one percept per action), which means the next action is not a function of the previous function and doesn't need to be remembered. Sequential means that the current decision COULD affect all future decisions/actions.

Static environments CANNOT change while the agent is taking its time deciding while dynamic environments CAN change and speed if important.

Agents in known environments know all outcomes (and probabilities) to its actions, essentially, the agent knows how the environment works. An unknown environment means the agent doesn't know all the details about the inner workings of the environment and some learning and exploration can be necessary.

(2.4): The Structure of Agents Agent = Architecture + Program Simple-reflex agents use condition-action rules; Model-based reflex agents keep track of the unobserved parts of the environment by maintaining an internal state; Goal-based reflex agents maintain the model of the world and goals to select decisions (that lead to the goal); Utility-based reflex agents maintain the model of the world and utility function to select PREFERRED decisions (that lead to the best expected utility: AVG(EU*p)).

State and Transition Representations

Atomic state representations have NO internal structure; Factored state representation includes fixed attributes (which can have values); and Structured states include objects and their relationships.

(2.5): Summary Go through the chapter summary.

Chapter 3: Solving Problems by Search

(3.1): Problem-Solving Agents Be comfortable defining a search problem.

(3.2): Example Problems

(3.3): Search Algorithms & Uniform Search Strategies Unlike A*, Greedy Best First Search is not always going to find the optimal solution, even with admissible heuristics. It is complete, not optimal, but often efficient.

Ignore sections 3.4.4 and 3.4.5 for the exam.

(3.5): Informed (Heuristic) Search Strategies Informed search relies on domain-specific knowledge / hints that help locate the goal state. $h(n) = h(\text{State n}) = $ relevant information about State n

You may be asked to solve a search problem by hand.

(3.6): Heuristic Functions Straight-line heuristics is admissible: it never overestimates the cost. An admissible heuristics is guaranteed to give you the optimal solution. Every consistent heuristics is admissible, but not the other way around. A dominating heuristic is a heuristic that estimates closer to the actual cost than another heuristic, with the offset that dominating heuristics are more costly to compute.

(3.7): A* The evaluation function for A* is $f(n) = g(\text{State}_n) + h(\text{State}_n)$ where $g(n)$ is the path cost from the initial node to node n, and $h(n)$ is the estimated cost of the best path that continues from node n to a goal node. A node n with minimum (or maximum if necessary) $f(n)$ should be chosen for expansion.

(3.8): Summary Go through the chapter summary. FOCUS ON A* algorithm

Chapter 4: Search in Complex Environments

(4.1): Local Search and Optimization Problems Local search doesn't care about the path to the goal, just getting to the goal. They're useful for pure optimization problems (finding the best state according to an objective function.) Generally use a single current state and generally move to neighbors of that state. Two key advantages are: little memory usage (usually a constant amount) and can find reasonable solutions in large of infinite (continuous) states spaces. The performance can be measured using completeness (guaranteed to find a solution when there is one and report when there isn't), cost-optimality (does it find a solution with the lowest path cost of all solutions), or time or space complexity.

Hill-climbing search

The most primitive informed search approach; it is a naive greedy algorithm and the objective function is the value of the next state. The agent can get stuck on peaks (local maxima), ridges (sequences of peaks), and plateaus (areas where the evaluation function has the same value).

Simulated Annealing

Accepts a move if it improves the objective value, and accepts some "bad" moves given some probability depending on the current objective value. Converges to a global optimum; in practice, it can give excellent results.

Evolutionary algorithms

Nature – speciation occurs when two similar reproducing beings evolve to become too dissimilar to share genetic information effectively or correctly. **Implementation** – speciation→some mathematical function that established the similarity between two candidate solutions in the population

. . . and everything related to Evolutionary algorithms that I covered in class (especially: EVERYTHING about GENETIC ALGORITHM) IGNORE TABU SEARCH

Genetic Programming (GP)

GP is an automated method for creating a working computer program from a high-level problem statement of a problem. It starts from a high-level statement of "what needs to be done" and automatically creates a computer program to solve the problem. Genotypes are trees, whereas the phenotypes (what can be seen) is the evaluation score. Mutation occurs by picking a node for random mutation (by generating a new random subtree) and replacing the node with the root of the new subtree. Crossover occurs by swapping the values of two nodes, without changing the positions or subtrees of those nodes (can be thought of as flipping the sign in an operation node).

Chapter 5: Adversarial Search and Games

(5.1): Game Theory I don't know what move my opponent will choose, but I am going to ASSUME that it is going to be the best / optimal option.

(5.2): Optimal Decision in Games

$$\text{MinMax}(n) = \begin{cases} \text{Utility}(n, MAX), if \ \text{Is-Terminal}(n) \\ \max_{a \in \text{Actions}(n)} \text{MinMax}(Result(n, a)), \\ \quad if \ \text{To-Move}(s) == MAX \\ \\ \min_{a \in \text{Actions}(n)} \text{MinMax}(Result(n, a)), \\ \quad if \ \text{To-Move}(s) == MIN \end{cases}$$

(5.3): Summary Go through the chapter summary.

Chapter 6: Constraint Satisfaction Problems

(6.1): Defining CSPs You may be asked to formally define a constraint satisfaction problem.

(6.2): Constraint Propagation: Inference in CSPs Minimum-remaining-values (MRV) heuristic: choose the variable with the "fewest" legal values

(6.3): Backtracking Search for CSPs Ignore sections 6.3.3 and 6.3.4.

(6.4): Summary Go through the chapter summary.

Chapter 7: Ant Colony Optimization Stigmergy – two
individuals interact indirectly when one of them modifies the environment and the other responds to the new environment at a later time.

(7.1): Pheromone Trail Individual ants deposit pheromones while travelling between nest and food source. Pheromone trail gradually evaporates over time. Pheromone trail strength accumulates with multiple ants using the path.

(7.2): Artificial Ant Properties **Memory** – the list of visited places (nodes in the graph). **Best fitness** – the lowest cost total distance traveled between places **Action** – select next location (node) to visit (and leave pheromones along the way) α – pheromone intensity weight β – heuristic weight

$$P(move \ i \rightarrow j) = \frac{d_{ij}^{\alpha} \times \frac{1}{h_{ij}}^{\beta}}{\sum_{k=1}^{N \ possible \ destinations} d_{ik}^{\alpha} \times \frac{1}{h_{ik}}^{\beta}}$$

Algorithm 0.1 Simulated Annealing

1: **function** Simulated-Annealing($problem$, $Schedule$) **returns** a solution state
2: $\quad current \leftarrow problem.INITIAL$
3: $\quad$ **for** $t = 1$ to ∞ **do**
4: $\quad\quad T \leftarrow$ Schedule(t)
5: $\quad\quad$ **if** $T == 0$ **then return** $current$
6: $\quad\quad$ **end if**
7: $\quad\quad next \leftarrow$ a randomly selected successor of $current$
8: $\quad\quad \Delta E \leftarrow$ Value($current$) $-$ Value($next$)
9: $\quad\quad$ **if** $\Delta E > 0$ **then** $current \leftarrow next$
10: $\quad\quad$ **else** $current \leftarrow next$ only with probability $e^{\Delta E/T}$
11: $\quad\quad$ **end if**
12: $\quad$ **end for**
13: **end function**

Algorithm 0.2 Hill-climbing search

1: **function** Hill-Climbing($problem$) **returns** a state that is a local maximum
2: $\quad current \leftarrow problem.INITIAL$
3: $\quad$ **while** true **do**
4: $\quad\quad neighbor \leftarrow$ a highest-valued successor state of $current$
5: $\quad\quad$ **if** Value($neighbor$) $\leq$ Value($current$) **then return** $current$
6: $\quad\quad$ **end if**
7: $\quad\quad current \leftarrow neighbor$
8: $\quad$ **end while**
9: **end function**

Algorithm 0.3 Best-First Search: A* Pseudocode

1: **function** Best-First-Search($problem, f$) **returns** a solution node or $failure$
2: $\quad node \leftarrow$ Node($State = problem.Initial$)
3: $\quad frontier \leftarrow$ a priority queue ordered by f, with $node$ as an element
4: $\quad reached \leftarrow$ a lookup table, with one entry with key $problem$.Initial and value $node$
5: $\quad$ **while not** Is-Empty($frontier$) **do**
6: $\quad\quad node \leftarrow$ Pop($frontier$)
7: $\quad\quad$ **if** $problem$.Is-Goal($node.State$) **then return** $node$
8: $\quad\quad$ **end if**
9: $\quad\quad$ **for all** $child$ in Expand($problem$, $node$) **do**
10: $\quad\quad\quad s \leftarrow child$.State
11: $\quad\quad\quad$ **if** s **not in** $reached$ or $child$.Path-Cost $<$ $reached[s]$.Path-Cost **then**
12: $\quad\quad\quad\quad reached[s] \leftarrow child$
13: $\quad\quad\quad\quad child$ to $frontier$
14: $\quad\quad\quad$ **end if**
15: $\quad\quad$ **end for**
16: $\quad$ **end while**
17: $\quad$ **return** $failure$
18: **end function**
19: **function** Expand($problem, node$) yields nodes
20: $\quad s \leftarrow node$.State
21: $\quad$ **for all** $action$ in $problem$.Actions(s) **do**
22: $\quad\quad s' \leftarrow problem$.Result($s, action$)
23: $\quad\quad cost \leftarrow node$.Path-Cost $+ problem$.Action-Cost($s, action, s'$)
24: $\quad\quad$ yield Node($State = s', Parent = node, Action = action, Path - Cost = cost$)
25: $\quad$ **end for**
26: **end function**

Algorithm 0.4 Genetic Algorithm Pseudocode

1: **function** Genetic-Algorithm($population$, $fitness$) **returns** an individual
2: $\quad$ **repeat**
3: $\quad\quad weights \leftarrow$ Weighted-By($population, fitness$)
4: $\quad\quad population2 \leftarrow$ empty list
5: $\quad\quad$ **for** $i = 1$ to Size($population$) **do**
6: $\quad\quad\quad parent1, parent2 \leftarrow$ Weighted-Random-Choices($populati$
7: $\quad\quad\quad child \leftarrow$ Reproduce($parent1$, $parent2$)
8: $\quad\quad\quad$ **if** small random probability **then** $child \leftarrow$ Mutate($child$)
9: $\quad\quad\quad$ **end if**
10: $\quad\quad\quad$ add $child$ to $population2$
11: $\quad\quad$ **end for**
12: $\quad\quad population \leftarrow population2$
13: $\quad$ **until** some individual is fit enough, or enough time has elapsed
14: $\quad$ **return** the best individual in $population$, according to $fitness$
15: **end function**
16: **function** Reproduce($parent1, parent2$) **returns** an individual
17: $\quad n \leftarrow$ Length($parent1$)
18: $\quad c \leftarrow$ random number from 1 to n
19: $\quad$ **return** Append(Substring($parent1$, 1, c), Substring($parent2$, $c + 1$, n))
20: **end function**