1 Introduction

Agents: Anything that can be viewed as *perceiving its environment through sensors* and *acting upon that environment through actuators*. The agent function maps from percept histories to actions.

Rational and Autonomous Agent:

Chooses an action that is *expected to maximize its performance measure* (see below), given the evidence provided by the *percept sequence* and *built-in knowledge*.

Autonomous means behavior determined by own experience (can learn and adapt).

PEAS Framework to specify AI's task: Performance measure (Objective criterion for success), Environment (takes input and delivers output here), Actuators (part of agent that delivers), Sensors.

Performance measure: Better to design according to what is wanted to be achieved rather than how (we think) it "should" behave.

Environment types:

- **Fully observable vs Partially**: Sensors can access **complete** state of the environment in each point in time. Complete refers to relevance, which depends on **performance measure**.
- **Deterministic vs Stochastic**: Next state of environment is completely determined by current state and action executed by the agent. Stochastic is only if model deals with probabilities. If deterministic except for actions of other agents then **Strategic**.
- **Episodic (Memoryless) vs Sequential**: Experience is divided into **atomic episodes**, and each episode depends on the episode itself. Sequential means current decisions can affect all future decisions.
- **Static vs Dynamic**: Environment is unchanged while the agent is deliberating. If environment doesn't change but agent's performance changes then **semi-dynamic**.
- **Descrete vs Continuous**: Limitd number of distinct, clearly defined percepts and actions.
- **Single vs Multi-agent**: Agent operating by itself. Gentle reminder that agent's performance measure depends on agent's behavior.

Structure of Agents:

- Table driven. Lookup table with percept sequence.
- Reflex agents: simple and model-based. Simple uses if-then, model uses transitional model (knowledge of world changes).
- Goal and utility-based. Goal picks action that brings it closer to goal, utility uses a score-function to internalise the perf.meas.; if u.f. and perf.meas. is aligned then agent is rational.
- Learning agent. Uses performance element, critic (feedback), learning element (making improvements), problem generator (generates new and informative experiences).

2 Uninformed Search

Uses only information available in the problem definition. Examples: **BFS** (expand minimum depth unexpanded node), **DFS** (expand deepest unexpanded node), **UCS** (expand least cost unexpanded node, Dijkstra stopped at popping goal; while Dijkstra finds distance from start to **every vertex**), **DLS** (DFS with depth limit), **IDS** (try for depth from 0 to ∞).

Criterion	BFS	UCS	DFS	DLS	IDS
Complete	Yes	Yes	No	No	Yes
Time	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimal	Yes	Yes	No	No	Yes

Complexities are measured in terms of: maximum branching factor b, depth of the least cost solution d, maximum depth of the state space m, maximum limit of depth l.

Note: Time of IDS is $(d+1)b^0 + db^1 + ... + b^d$. Also, DLS may visit extraneous nodes even if no loops (example 1 missionary and 1 cannibal). Overhead: (node IDS - node DLS)/node DLS.

3 Informed Search

Best-first Search: Use an evaluation function $f(n)$ for each node to estimate desirability. Expand most desirable unexpanded node. Special cases:

- **Greedy**: Uses $f(n) = h(n)$. Not complete or optimal, time/space complexity $O(b^m)$ but good heuristic can dramatically improve.
- **A***: Use $f(n) = dist(n) + h(n)$. Complete unless infinitely many nodes with $f \leq f(G)$. Time/space exponential. Optimal using tree search (no memoization) if h is admissible, graph search (memoization) if h consistent (as $f(n)$ is non-decreasing along any path). Variations:
  - **IDA*/Iterative Deepening A***: Cutoff based on f value (instead of depth). Remember best value of f that was cutoff. Space $O(bm)$.
  - **SMA*/Simplified Memory-bounded A***: If memory is full, throw away worst f value node.
- Remember that the key idea is **pruning** (Ben Leong, during lecture).

Admissible Heuristic: Obeys $h(n) \leq h^*(n)$ (i.e. always less than actual cost).

Consistent Heuristic: $h(n) \leq c(n,a,n') + h(n')$, where $c(n,a,n')$ denotes cost to move from state n to n' using "edge/link" a. Consistent implies admissible if $h(n) = 0 => n$ is goal.

Dominance: If $h_2(n) \geq h_1(n)$ then h_2 is better for search.

Local Search: Where path is irrelevant, the goal state itself is the answer.

- **Hill Climbing**: Keep going to best neighbour state.
- **Simulated Annealing**: Select random neighbour. If better then go there, otherwise (still) go there, probability $e^{\Delta f/T}$. Gradually decrease T (time representation) with time. Δf large or T small imply less probability to go to worse state.
- **Beam Search**: Keep best k states in hill climbing.
- **Random Restarts**: Restart after find a minima for a possibly lower one.
- **Genetic Algorithms**: fitness function as evaluation, produce next states by selection, crossover, mutation.

4 Adversarial Search

α-β **Pruning**:

- α is the best alternative for MAX player.
- β is the best alternative for MIN player.
- My own understanding: alpha-beta has 3 + 1 rules.
- Zero (**Minimax with 2 extra counters**). Keep 3 values: n, α, β. n stands for node. Update n according to minimax.
- First (**backtracking**). If parent is MAX, $\alpha(\text{PARENT}) := n(\text{PARENT}) := \max(n(\text{CHILD}), \text{init } \alpha(\text{PARENT}))$. Similar for MIN.
- Second (**heritage passing**). If parent is MAX, $\alpha(\text{CHILD}) := \max(\alpha(\text{PARENT}), \text{init } \alpha(\text{child}))$. Essentially swap all PARENT with CHILD and vice versa in First Rule.
- Similarly for MIN. This can be applied recursively.
- Third (**pruning**). A **link** is **pruned** (*thrown away* or not explored) if its parent's α is $\geq$ to its β.
- Then return the node value of the topmost vertex for next best move.

Machine Learning (probably not important): A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E – Tom Mitchell.

Types of Feedback:

- **Supervised**: Correct answer is given for each example.
- **Unsupervised**: No answers given.
- **Weakly Supervised**: Correct answer given but not precise.
- **Reinforcement**: Occasional rewards and punishments given.

Classes of Supervised Learning:

- **Regression**: Output (predict results) continuously, map input variables to some continuous function.
- **Classification**: Outputs discretely, map input variables into discrete categories.

Entropy: For a random variable X entropy is $E[-\log_2 p(X)] = -\sum_x p(x) \log_2 p(x)$.

Information Gain: Entropy of node − weighted average of entropy of child nodes (based on how many data points there are on child node). Can be negative.

5 Learning Decision Trees

Choosing Attribute Tests: Learn from examples, that is simpler than *true* decision trees (which only exists because we train a model). Algo is recursive DnC (divide-and-conquer).

- If remaining examples is only T/F, return that value.
- If there is a mix, choose highest-importance attribute to split, then recurse.
- If there are no attributes left for split, then we return most common value of current examples.
- Note from TA: call this node **impure**. If cannot choose (equal number of T/F) call it **undecided**. Call the situation **missing attributes or inconsistent data**.
- (Rare). If there are no examples left, return most common value of parent's examples.

Child Entropy: If parent has N nodes (entropy of parent is not multiplied by any weight), children have $C_1, C_2, \ldots, C_k$ nodes, total child entropy is

$$\sum_{i=1}^{k} \frac{C_i}{N} \left(\sum_{x \in X_i} p(x) \log_2 \left(\frac{1}{(p(x)} \right) \right),$$

with X_i denoting child i's random variable.

Pruning: Overfitting becomes likely as number of attributes/features (n) grows, and less likely as we increase number of training examples (m).

Do decision tree pruning: throwing away irrelevant nodes. We replace with leaf nodes in that case, and we repeat.

Note: Minimum sample pruning: **each node**'s sample must be at least threshold, **not** parent node's sample.

6 Uncategorized Extras (but important nonetheless)

Pre-computation of Entropy: Reduce constant factor of calculating $I(x,*) := -x\ \log_2(x) - (1-x)\ \log_2(1-x)$, where $* := 1-x$. $I(x,*)$ is concave on $[0,1]$. It is increasing on $0 \le x \le \frac{1}{2}$ and decreasing on $\frac{1}{2} \le x \le 1$; also, $I(x,*) = I(1-x,*)$.

Concrete Values for b at most 10. $I\left(\frac{1}{2},*\right) = 1.0000$, $I\left(\frac{1}{3},*\right) = 0.9183$, $I\left(\frac{1}{4},*\right) = 0.8113$,

$I\left(\frac{1}{5},*\right) = 0.7220$, $I\left(\frac{2}{5},*\right) = 0.9710$,

$I\left(\frac{1}{6},*\right) = 0.6500$,

$I\left(\frac{1}{7},*\right) = 0.5917$, $I\left(\frac{2}{7},*\right) = 0.8631$, $I\left(\frac{3}{7},*\right) = 0.9852$,

$I\left(\frac{1}{8},*\right) = 0.5436$, $I\left(\frac{3}{8},*\right) = 0.9544$,

$I\left(\frac{1}{9},*\right) = 0.5033$, $I\left(\frac{2}{9},*\right) = 0.7642$, $I\left(\frac{4}{9},*\right) = 0.9911$,

$I\left(\frac{1}{10},*\right) = 0.4690$, $I\left(\frac{3}{10},*\right) = 0.8813$.

Note: I do not put until $b \le 15$ due to "not wanting to overfit". I think it's best to just compute manually for large enough b, to "double check" your understanding (under exam pressure).

Simplified Way to Count Entropy:

$$\begin{aligned} I\left(\frac{a}{b},*\right) &= \frac{a}{b} \log\left(\frac{b}{a}\right) + \frac{b-a}{b} \log\left(\frac{b}{b-a}\right) \\ &= \frac{a\log(b) - a\log(a) + (b-a)\log(b) - (b-a)\log(b-a)}{b} \\ &= \frac{b\log(b) - a\log(a) - (b-a)\log(b-a)}{b}. \end{aligned}$$

Pre-computing Individual Logs, values less than 16: Define $a := \log_2(2) = 1.0000$, $b := \log_2(3) = \mathbf{1.5850}$, $c := \log_2(5) = \mathbf{2.3219}$, $d := \log_2(7) = \mathbf{2.8074}$, $e := \log_2(11) = \mathbf{3.4594}$, $f := \log_2(13) = \mathbf{3.7004}$.

Number	Decomposed	Value
4	$2a$	2.0000
6	$a+b$	2.5850
8	$3a$	3.0000
9	$2b$	3.1699
10	$a+c$	3.3219
12	$2a+b$	3.5850
14	$a+d$	3.8074
15	$b+c$	3.9069

My Mistakes that is not covered in Sectional Notes:

- Always put the **complete** State Representation + Implicit Assumptions + Rigorous Mathematical Transition Functions (with **ALL** constraints and possible actions — including ones that will create exceptions) + Representation Invariants (if needed).
- Remember that $I\left(\frac{3}{7},\frac{1}{7},\frac{3}{7}\right) = \frac{3}{7}\log\left(\frac{7}{3}\right) + \frac{1}{7}\log(7) + \frac{3}{7}\log\left(\frac{7}{3}\right)$.
- Case against overfitting: may take in errors, unnecessary and hard-to-humanly-understand models, taking in irrelevant extra parameters.
- Case against underfitting: many ways to form short hypotheses, can be done in multiple ways/representations, not enough parameters/not representative of whole data.
- For Lookup-table stuff, always remember to use DP for faster game tree! (Utilize previous questions, too.) [from 2007 midterm]

Pre-thought Answer to the 3 Free-Point Question:

- Do big things first. This mod is hard, and should be prioritized. It is also very useful in current job market, so must understand. I learnt about prioritizing and modifying my **internal** utility function.
- Ask for help (from TAs, friends that are better than you, even from Prof). You will have blind spots regardless of how good you are (and/or how good you think you are). Asking for help makes your understanding much more rigorous, and often times helps the person you are asking help from, too.
- Friends and environments are important. You will work with humans in actual jobs. We cannot teach intuition to AI (at least, not yet), they can only serve as tools; so it is not wise to rely your progress on a *tool* (whatever the tool is: a book, the internet, even your own knowledge). Quoting Prof Ben, **AI can do things right, but only humans can do the right thing**.

7 Regression

Gradient Descent: $x^{(i)}, y^{(i)} = i$-th sample, $X_{m\times n}$ = all samples stacked vertically, Y similarly. Squared loss function:

$$J(w) = \frac{1}{2m} \sum_{i=1}^{m} (h_w(x^{(i)}) - y^{(i)})^2$$

$$\frac{\partial J(w)}{\partial w_j} = \frac{1}{m} \sum_{i=1}^{m} (h_w(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$$

Repeat $w_j := w_j - \alpha \frac{\partial J(w)}{\partial w_j}$.

Mean Normalization: Replace each feature x_i with $\frac{x_i - \mu_i}{\sigma_i}$.

Normal Equation: Least squares solution to $Xw = Y$ is $w = (X^T X)^{-1} X^T Y$.

8 Logistic Regression

Improving Decision Tree:

- **Pruning**: Prevent node from being split when it fails to cleanly separate examples. Check relevence of an attribute by statistical tests.
- **Continuous valued attributes**: Convert into discrete intervals.

Logistic Regression: Logistic function is $Logistic(z) = \frac{1}{1+e^{-z}}$. Hypothesis is $h_w(x) = \frac{1}{1+e^{-w\cdot x}}$. Interpret as: $h_w(x)$ = probability that y is 1 on input x. Cost function is:

$$\begin{aligned} \textsc{Cost}(h_w(x), y) &= \begin{cases} -\ln h_w(x) & \text{if } y = 1 \\ -\ln(1 - h_w(x)) & \text{if } y = 0 \end{cases} \\ &= -y \ln h_w(x) - (1-y)\ln(1 - h_w(x)) \end{aligned}$$

and loss function is average of cost over all samples. Update is same as normal gradient descent: repeat $w_j := w_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_w(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$.

Multi-class Classification: Train logistic classifier for each class. Predict the one that gives maximum probability.