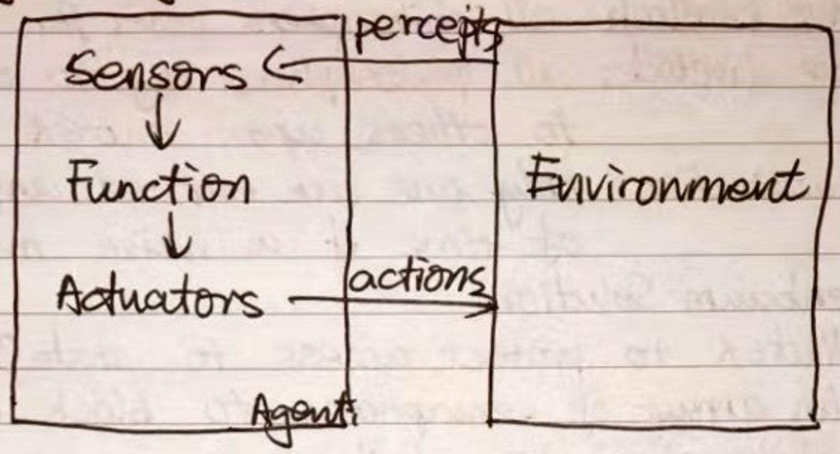
CS2109S Notes.

Intelligent Agents:



The agent function maps from perception histories to actions.

$$[f: \mathcal{P}^* \to \mathcal{A}].$$

Agent = (physical) architecture + program

Rational Agents.

A rational agent will choose an action that is <u>expected</u> to maximize its performance measure. given to percept sequence + built-in knowledge.
Rationality ≠ omniscience.
An agent is autonomous if its behavior is determined by its own experience (can explore and learn).

Defining the Problem: The PEAS Model / Framework.

- Performance Measure
- Environment
- Actuators
- Sensors.

Date No.

Environment:

- Fully Observable VS Partially Observable.
- Deterministic VS Stochastic.
  Former: The next state is fully determined by the current state and the agent's action.
  Strategic Environment: deterministic except for actions of other agents.
- Episodic VS Sequential.
  The choice of action in each episode depends only on the episode itself. ("Memoryless")
- Static VS Dynamic:
  The state does not change while the agent is deliberating.
  Semi-Dynamic: The state does not change, but the performance score does.
- Discrete VS Continuous
  A limited number of distinct, clearly defined percepts and actions.
- Single Agent VS Multi-Agent.

Models for Agent Organization.

Complexity Increases.

- Simple Reflex Agent.
  What the world is like → Actions.
- Model-Based Reflex Agent:
  Previous + evolution of the world + effects of actions → actions.
- Goal-Based Agent
  Previous + potential next states (+ possibly a sequence to the goal) → actions.
- Utility-Based Agent.
  Previous + how happy I will be in such a state → actions.
- Learning Agent.
  Learn, analyse performance, actively explore and improve.
  May not maximise short-term performance.

Date No.

Exploitation VS Exploration.

VS
- Maximising its expected utility according to current knowledge.
- Trying to learn more about the world.

Different Problem Types:

Deterministic, fully observable → Single state.
Non-observable → sensorless problem.
Non-deterministic/partially observable → contingency
Unknown state space → exploration problem.

We use Graph to solve single state problems.

Tree Search:

Offline, simulated exploration of state space by generating successors of already-explored states. (Expanding states).
We use frontier/fringe.

State VS Note

State: physical configuration.
Node: state, parent, action, path cost, depth.

Evaluation of Search Strategies. (COST).

Completeness: always find a solution?
Optimality: always find a least cost solution?
Space Complexity } $b :=$ branch factor
Time Complexity } $d :=$ depth of optimal solution
$m :=$ max depth of search space

Uninformed Search Strategies.

Breath-First Search. (BFS)
Uniform-cost Search (UCS)
Depth-First Search (DFS)
Depth-Limited Search (DLS)
Iterative Deppening Search (IDS).

	BFS	UCS ②	DFS	DLS	IDS ⑤
Complete	Yes	Yes	No	No	Yes
Time	$O(b^{d+1})$	③ $O(b^{\lceil C^*/\varepsilon \rceil})$	$O(b^m)$ ④	$O(b^l)$	$O(b^{d})$ ⑥
Space	$O(b^{d+1})$ ①	$O(b^{\lceil C^*/\varepsilon \rceil})$	$O(bm)$	$O(bl)$	$O(bd)$
Optimal	Yes	Yes	No	No	Yes.

① All nodes at the lowest level (leaves).
② Not necessarily a variation of BFS. Eg. Dijkstra's Algo.
③ ε is the lower bound of all step costs, so C^*/ε is the upper bound of # steps to reach the optimal sol.
④ Terrible if m is large. Can be much faster than BFS if the solutions are dense.
⑤ A variation of DFS, so $O(bd)$ Space.
⑥ $N_{IDS} = (d+1)b^0 + db^1 + (d-1)b^2 + \cdots + 3b^{d-2} + 2b^{d-1} + 1b^d$
$N_{DFS} = b^0 + b^1 + b^2 + \cdots + b^d$.
Overhead $= \frac{N_{IDS} - N_{DFS}}{N_{DFS}}$. The divisor is the true cost.

"Optimality"
Generally refers to the solution itself, that is, whether the generated solution solves the problem at the least cost; instead of the process of generating the solution.

Tree Search + Memoization = Graph Search.

Bidirectional Search.

$2O(b^{d/2}) \ll O(b^d)$.

Operations must be reversible: $pred(succ(u)) = succ(pred(u)) = u$

Uninformed Search vs Informed Search.

Uninformed Search: Has no info about the goal state and where it "might" be. Tries to search in the whole space exhaustively.

Informed Search: Has some idea about how close the goal state is. Usually makes uses of heuristic/estimate/evaluation.

Informed Search Strategies.

- Best-First Search
  - Greedy Best-First Search
  - A* Search
  - Memory-Bounded Heuristic Search
    - Iterative Deepening A* (IDA*).
    - Recursive Best-First Search (RBFS)
    - Simplified Memory-Bounded A* (SMA*).

Uniform-Cost Search cares about the cost so far, whereas Greedy Best-First Search cares about the current "distance" to the goal.

A* search cares about both. UCS + GBFS ≈ A*.

Date No.

A* Search.

Idea: Avoid expanding paths that are already expensive, while taking into account the distance to the goal state.

Evaluation Function:

$$f(n) = \underbrace{g(n)}_{\text{cost}} + \underbrace{h(n)}_{\text{heuristic}}.$$

Admissible Heuristic: → use tree search to get optimal solution

For all n, $h(n) \le h^*(n)$

Never over-estimates → conservative est.

Consistent Heuristic: → use graph search

For all n, n', $h(n) \le c(n,a,n') + h(n')$

$$\Rightarrow f(n) = g(n) + h(n)$$
$$\le g(n) + c(n,a,n') + h(n')$$
$$= g(n') + h(n') = f(n').$$

∴ $f(n)$ is a non-decreasing sequence.

Proof of optimality:

① Admissible:

Starting from equal h. The difference comes from different g. The diff in order of expansion is guaranteed by $\le h^*(n)$

∴ $f(n) \le f(G) < f(G_2)$.

② Consistent:

The expanded region adds f-contours gradually → i.e., coming back from outside of the region to an internal node definitely incurs at least the cost so far.

Dominance.

$h_2(n) \ge h_1(n)$ for all n → h_2 dominates h_1.

Dominance has <u>nothing</u> to do with admissibility

Consistent $\Longrightarrow$ Admissible
$\not\Longleftarrow$

Date No.

Relaxed Problem
A problem with fewer restrictions.
The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem.

RBFS:
keep track of the 2^{nd} best f-value seen thus far.
Backtrack → to the 2^{nd} best when we end up at a node worse than 2^{nd} best f-value.

Pruning

IDA*:
At each iteration, perform a DFS, cutting off a branch when its total cost $f(n)$ exceeds a threshold. The threshold will increase.

SMA*:
When memory is full, drop the worst f-value. Update the parent of best path on dropped sub-tree.

Local Search
For optimization. Unlike informed search, which expands new candidate solutions.
Local search improves and existing solution.

Types of Local Search.
- Hill-Climbing Search
- Simulated Annealing Search
- Beam Search: Perform K Hill-Climbing Search in parallel
  - Local Beam Search: k threads share info.
  - Stochastic Beam Search: k threads are independent.
- Genetic Algorithms:
  Population: k randomly generated initial states

Fitness Function: Higher values for better states
How to evolve: selection, crossover, mutation

Formal Description of Finite Two-Person Games of Perfect Information in which the Two Players Move Alternatively.
- An Initial State
- Successor Function
- Terminal Test
- Utility Function : MAX player maximises it.

Minimax Algorithm:
Alternatively pick min and max at each level assuming the opponent is also optimal. For MAX node we pick the max of its children.

Alpha-Beta Pruning (α-β Pruning).
(α, β) is a range. If it becomes an empty set or a single number, we back-track.
Order matters!
Properties:
- Still optimal.
- Perfect ordering $\Rightarrow O(b^{m/2}) \Rightarrow$ doubles depth of search.

Imagine all good moves are on the LHS, so it becomes a DFS that can go twice the depth in the same amount of time.

Summary:
At MIN\MAX node, can stop if we find a node that is smaller\larger or equal to $\alpha \backslash \beta$, which will give $\beta \backslash \alpha$ smaller\larger or equal to $\alpha \backslash \beta$.

Resource Limit: Solutions
- Cutoff Test (e.g. depth limit). } Standard
- Better Evaluation Function. } Standard
- Memoization & Transpositions (diff sequences give same state)
- Pre-Computation of Opening/Closing Moves.

Date No.

Cutting-Off Search
Terminal Test is replaced by cutoff threshold.
Utility is replaced by Eval. (an estimate of util)

Machine Learning.
ETP:
A computer program is said to learn from experience E w.r.t. some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

Problem Solving:
- known exact formula: → Direct Coding.
- Possess solutions to related problem → Search.
- Ask experts → Expert System.
- collect data → ML.

Types of Feedback:
- Supervised: Correct answer given for each example.
- Unsupervised: No answers given.
- Weakly Supervised; Correct answer given, not precise.
- Reinforcement: Occasional rewards given: e.g. robot navigating a maze.

Major Classes of Supervised Learning.
- Regression: (Continuous output → (f: input → continuous function))
- Classification: (Discrete output → (f: input → discrete categories)).

Hypothesis.
The proposed function f such that $f(x) = y$ for all examples (x, y).

NUS National University of Singapore

Date No.

Hypothesis Space

The range from which we choose the hypothesis.
n boolean attributes give 2^{2^n} truth tables, i.e., 2^{2^n} functions in the space.

Decision Tree Learning (DTL).

Preferably compact.
Use majority rule for missing attributes.
(can be used for regression tasks).

Information Theory

Information Content (Entropy) when the value of an attr is known.

$I(P(v_1), \cdots, P(v_n))$
$= -\sum_{i=1}^{n} P(v_i) \lg P(v_i)$, where v_i is the i-th value.

Special case: p positive examples and n negative examples.

$$I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) = -\frac{p}{p+n} \lg\left(\frac{p}{p+n}\right) - \frac{n}{p+n} \lg\left(\frac{n}{p+n}\right).$$

$$\text{remainder}(A) = \sum_{i=1}^{V}\left(\frac{p_i+n_i}{p+n} \cdot I\left(\frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i}\right)\right).$$

Intuition: prob of getting the i-th subset times the entropy gain from this subset.

Information Gain:

$$IG(A) = I(A) - \text{remainder}(A).$$

Inductive Learning:

Learn a function from examples.

Occam's Razor: prefer the simplest hypothesis consistent with data → avoid over-fitting.

Date No.

Linear Regression: ($\Rightarrow$ Convex Function: existence of global min).

Cost Function: Mean Squared Error. (MSE).

$$J(\underset{\sim}{w}) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_{\underset{\sim}{w}}(\underset{\sim}{x}^{(i)}) - \underset{\sim}{y}^{(i)}\right)^2.$$

where $\underset{\sim}{w}$ is the coefficient vector

$\underset{\sim}{x}^{(i)}$ is the i-th input example.

$\underset{\sim}{y}^{(i)}$ is the i-th output example.

m is the # coefficients.

Remarks: If bias is concened, increment m by 1 and augment $\underset{\sim}{x}^{(i)}[0]$ with 1. $\underset{\sim}{w}[0]$ should be the bias.

Cost Function: Mean Absolute Error (MAE).

$$J(\underset{\sim}{w}) = \frac{1}{m} \sum_{i=1}^{m} \left| h_{\underset{\sim}{w}}(\underset{\sim}{x}^{(i)}) - \underset{\sim}{y}^{(i)} \right|.$$

Remarks: Robust against outliers.

Not differentiable at $h_{\underset{\sim}{w}}(x^{(i)}) - y^{(i)} = 0$

Gradient Descent:

$$w[i] := w[i] - \alpha \frac{1}{m} \sum_{j=1}^{m} \left(h_{\underset{\sim}{w}}(\underset{\sim}{x}^{(j)}) - \underset{\sim}{y}^{(j)}\right) \underset{\sim}{x}_j[i]$$

where α is the learning rate.

Must be updated simultaneously.

Variants of GD:

- (Batch GD): Consider all training examples
- Stochastic GD:
  - Consider one data point at a time
  - Cheaper / faster.
  - More randomness → might escape local min
- Mini Batch GD: A mixture of both: Consider a subset of examples.

Feature Scaling:

GD does not work very well if features have significantly different scales → Make them vary roughly at the same scales.

Mean Normalization:

$$x_i \longleftarrow \frac{x_i - \mu_i}{\sigma_i}.$$

$$\mu_i = \frac{1}{N}\sum_{j=1}^{N} x_i^{(j)}. \qquad \sigma_i = \sqrt{\frac{1}{N}\sum_{j=1}^{N}\left(x_i^{(j)} - \mu_i\right)^2}$$

Polynomial Regression:

$$x_1 \leftarrow x, \quad x_2 \leftarrow x^2, \quad x_3 \leftarrow x^3 \quad \cdots$$

Need to do feature scaling.

Normal Equation.

m features and N examples.

$$\underset{\sim}{x}^{(i)} = [x_0^{(i)} \quad x_1^{(i)} \quad \cdots \quad x_m^{(i)}].$$

$$X = \begin{bmatrix} \underset{\sim}{x}^{(1)} \\ \underset{\sim}{x}^{(2)} \\ \underset{\sim}{x}^{(3)} \\ \vdots \\ \underset{\sim}{x}^{(N)} \end{bmatrix}$$

can add a bias column s.t. $X =: [1 \ X]$.

$$Y = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(N)} \end{bmatrix}$$

$$\therefore \quad X\underset{\sim}{w} = Y$$

$$\Rightarrow \quad X^T X \underset{\sim}{w} = X^T Y$$

$$\Rightarrow \quad \underset{\sim}{w} = (X^T X)^{-1} X^T Y.$$

$X^T X$ must be invertible ⇒ no duplicate examples

Comparison between GD and NE.

GD	NE
• Need to choose α	• No α.
• Many iterations	• One shot.
• Works well even when N is large.	• $X^T X$ must be invertible.
	• slow if N is large. $O(N^3)$.