# Chapter 2

## Rational Agent vs Omniscience Agent

A ***Rational agent*** carries out an action with the best possible outcome given some relevant information and is expected to maximize its performance measure. An ***Omniscience agent*** knows the actual outcome of its actions and it’s garneted to be correct at each decision; However, omniscience is impossible in reality.

### Requirements of Rational Agent

* Information gathering or exploration
* Learning: Agent gets modified or augmented as it gains experience.
* Autonomy: Agent is autonomous if its behavior is based only on its own experience.

## Agent Definition

An ***agent*** is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators.

## Agent function vs Agent Program vs Architecture

***Architecture***is the machinery that the agent executes on. ***Agent Program*** is a concrete implementation of an agent function. ***Agent function*** is an abstract mathematical mapping from the percept sequence (history of all the agent has perceived till now) to an action.

***Agent function*** cannot store previous percepts anywhere; However, ***Agent Program*** is cable if doing so.

Agent = Architecture + Agent Program

## Performance Measure

***Performance measure*** evaluates a sequence of environment states and depends ONLY on environment states and not the agent states.

## Task Environments (PEAS)

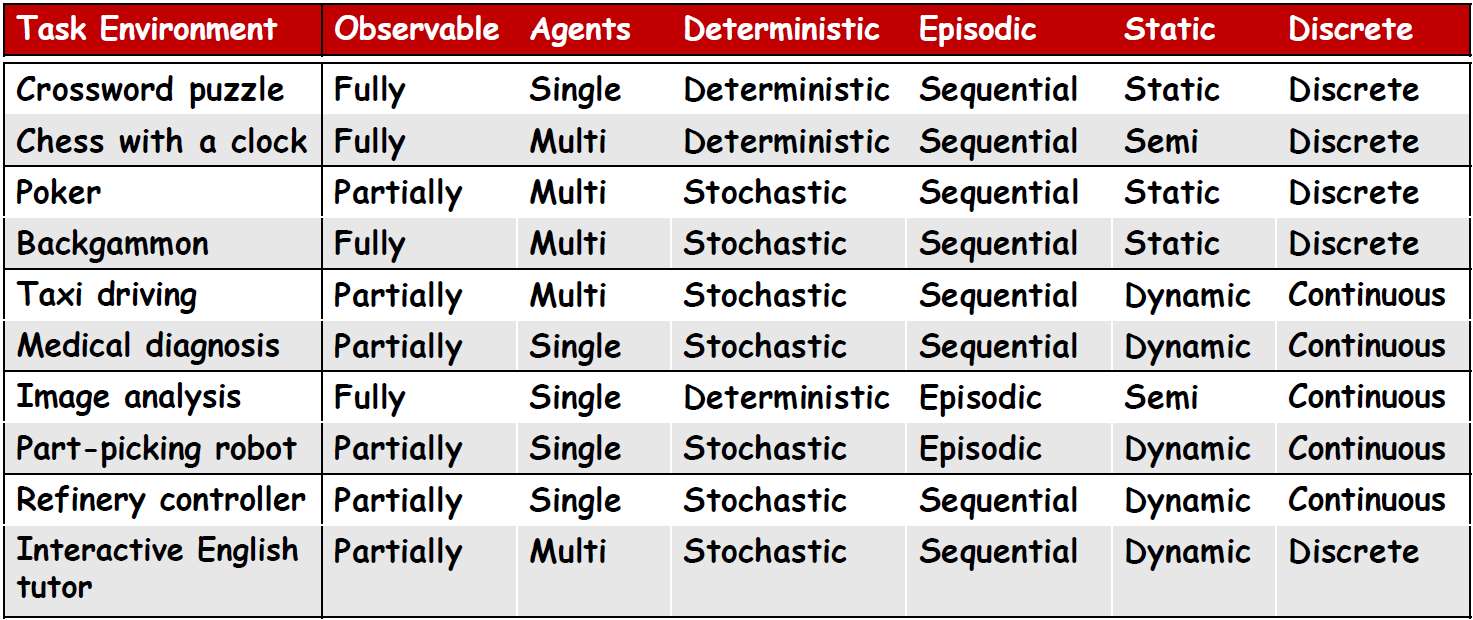
* ***Performance measure*** (Car: Safety, speed, legal, comfortable, …)
* ***Environment*** (Car: Other cars, road, passengers, …)
* ***Actuators*** (Car: Wheel, brake, horn, display, …)
* ***Sensors*** (Car: GPS, speedometer, cameras, accelerometer, engine sensors, …)

## Environment Types

* ***Fully Observable*** vs ***Partially Observable***: How vast can the agent observe its environment.
* ***Single-Agent*** vs ***Multi-Agent***: Number of agents are involved in the task. Multi-Agents can be competitive or cooperative.
* ***Deterministic*** vs ***Stochastic***: When next state can be completely determined by the current state.
* ***Discrete*** vs ***Continues***: Environment is discrete only if there are finite number of states at hand, its otherwise continues.

***Episodic*** vs ***Sequential***: Episodic when the choice of action in each episode depends only on the episode itself.

* ***Static*** *vs* ***Dynamic*** *vs* ***Semi-Dynamic***: Environment is static when it’s unchanged while agents running; Otherwise, it’s dynamic. Environment can be semi-dynamic when environment only changes by the agent’s actions.
* ***Known*** vs ***Unknown***: Environment is known only when the outcomes or outcome probabilities for all actions are given.

******

## Agent Program Types

* ***Table-Driven Agent***: A lookup table is used to map every possible percept sequence to the corresponding action. Most efficient implementation, but at the cost of humongous space usage.
* ***Simple Reflex Agent***: These types ignore the percept history and act only on the basis of the current percept. The agent function is condition-action rule based. Infinite loops are often un-avoidable and very limited intelligence.
* ***Model Based Reflex Agent:*** These types are mostly partially observable and they update their internal state information based on Information about how the world evolves and How the agent’s actions effect the world. They use their prior knowledge to decide for new percepts.
* ***Goal Based Agents***: These types are better than *Model-Based-Reflex-Agents* as they consider for future on each action. They may be less efficient but highly flexible.
* ***Utility Based Agent***: These agents care how the goal is reached by their internal utility function. There are useful when either when there are conflicting goals, only some of which can be achieved, the utility function specifies the appropriate tradeoff or there are several goals to aim for.
* ***Learning Based Agent***: These agents improve themselves on the experience they gain.
  + Learning element: Responsible for making improvements by learning.
  + Critic: Describes how well the agent is doing and feedbacks the learning element.
  + Performance Element: Responsible for selecting external actions.
  + Problem Generator: Suggests actions that may lead to new and informative experience.

## Representing States

* ***Atomic Representation***: Has no internal structure.
* ***Factored Representation***: Splits each state into variables and attributes.
* ***Structured Representation***: Objects and their various and varying relationships can be described explicitly.

# Chapter 3

## Fringe (Frontier)

**Fringe** is the set of all search nodes that haven’t been explored yet and is implemented as a priority queue (insert and remove).

## Evaluation

* ***Branching Factor***: Maximum number of successors of any state. Shown as **b**.
* ***Shallowest Goal node***: Minimal length (≠ cost) of a path between the initial and a goal state. Shown as **d**.

## Search Performance Measure

* ***Completeness***: A search algorithm is complete only if it can find a solution whenever one exists. (Not complete if there are infinite number of states or there are finite number of states but we don’t discard visited nodes at their revisit state)
* ***Optimality***: Optimality means returning a minimum cost path whenever a solution exists.
* ***Complexity***: It measures time and amount of memory consumption by the algorithm.

## Blind vs Heuristic Strategies

* ***Blind (Un-informed)***: They only exploit the positions of the nodes in the fringe regardless of their priority.
* ***Heuristic (Informed)***: Strategies exploit state descriptions to order fringe (Most promising nodes are stored at the beginning)

## Breadth First Search (BFS)

* ***Strategy***: Blind search (un-informed)
* ***How***: New nodes are inserted at the end of the fringe.
* ***Arc Cost***: 1
* ***Cons***:
  + If problem has no solutions, BFS might run for ever.
  + High space complexity.

## Depth First Search (DFS)

* ***Strategy***: Blind search (un-informed)
* ***How***: New nodes are inserted at the beginning of the fringe.
* ***Arc Cost***: 1
* ***Cons***:
  + Neither complete, nor optimal
* ***Upgrades:***
  + Avoid revisited nodes:
    - Store paths in the visited list. (Only avoids loops)
    - Store all generated states in the visited list. (Same space complexity as BFS)

## Depth Limited Search (DLS)

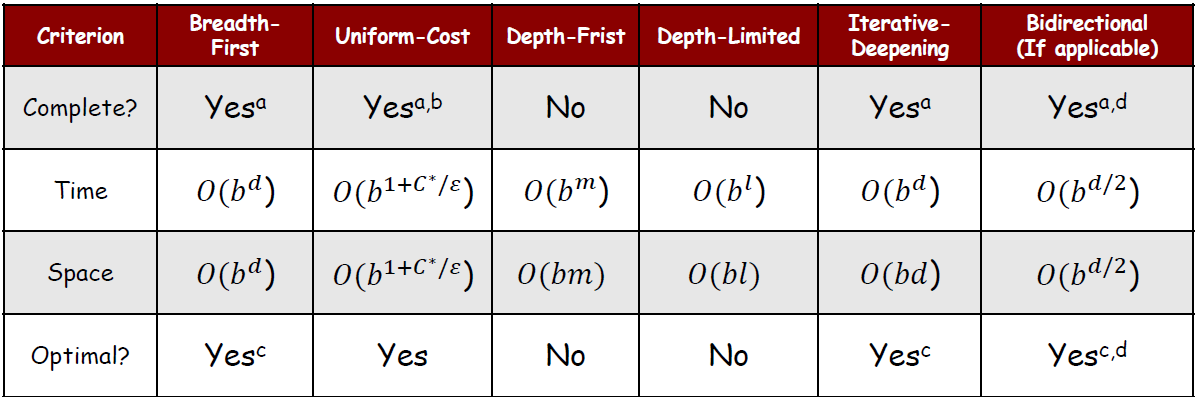
* ***Strategy***: Blind search (un-informed)
* ***How***: New nodes are inserted at the beginning of the fringe.
* ***Outcomes***
  + Solution
  + Failure (No solution)
  + Cutoff (No solution within cutoff)
* ***Arc Cost***: 1

## Iterative Deepening Search (IDS)

* ***Strategy***: Blind search (un-informed)
* ***How***: Increase depth search at each cycle.
* ***Arc Cost***: 1
* ***Pros***:
  + IDS is the preferred method when search space is large and depth is unknown.
  + IDS has the same space complexity as BFS and same time complexity as DFS.

## Uniform Cost Search (UCS)

* ***Strategy***: Blind search (un-informed)
* ***How***: Nodes in fringe are stored in increasing g(n) where g(n) is
* ***Arc Cost***: C(action) ≥ ɕ > 0
* ***Upgrades:***
  + Avoid revisited nodes: Store expanded nodes into closed list.



## Problem-Solving-Agent algorithm

* ***State Space***: Each state is a discrete and abstract representation of collection of states that can be finite or infinite. (8 Queens: All arrangements of 0, 1, …, 8 on board)
* ***Successor Function***: Successor function implicitly represents all the actions that are feasible in each state and only results of actions and their costs are returned. (8 Queens: Each of the successors is obtained by adding one queen in an empty square)
* ***Path Cost***: Cost of all arcs in the path. (8 Queens: Irrelevant)
* ***Goal State***: The goal that may be described directly, partially or by a condition. (8 Queens: 8 queens without threatening each other)

## Assumptions in Basic Search

World is static, discreditable, observable and actions are deterministic.

## Best First Search

* ***Strategy***: Blind search (un-informed)
* ***How***: Sorts the fringe in increasing f(N) where f(N) is the estimated cost (The smaller the better)
* ***Heuristic***
  + Cost of a solution path through N (f(N) = g(N) + h(N))
    - g(N): Cost of the path from **Start** to node **N**.
    - h(N): Cost of the path from **N** to **Goal**.
  + Cost of the path from **N** to **Goal** (f(N) = h(N)) => Greedy best-search

## Heuristic Function

The heuristic function h(N) ≥ 0 estimates the cost to go from state **N** to a **Goal** state.

* ***Euclidean distance***:
* ***Manhattan distance***:
* f(N) = g(N) + h(N)
* g(N): Cost of the path from **Start** to node **N**.
* h(N): Admissible heuristic cost of the path from **N** to **Goal**.

## Admissible vs Consistent Heuristic

If we consider to be the cost for the optimal path from **N** to the **Goal**, heuristic is **admissible** if:

()

**Note:** An admissible heuristic function is always optimistic!

An admissible heuristic h is **consistent** if for each node **N** and each child **N’** of **N**:

**Note:** If **h** is consistent, then whenever **A\*** expands a node, it has already found an optimal path to this node’s state.

## A\* Search (Most popular AI algorithm)

* ***f(N) = g(N) + h(N)***: h(N) is the admissible heuristic cost of the path from **N** to **Goal** and g(N) is the cost of the path from **Start** to node **N**
* ***Pros***
  + Complete & optimal
* ***Cons***
  + A\* runs for ever if no solutions exist. (Time-limit is given to avoid this)
  + Keeps all explored nodes in the memory.
* ***Upgrades***
  + We can discard a node revisiting a state if the cost of the new path to this state is ≥ cost of the previous path by storing it in a closed list.

## Heuristic Accuracy

Let be two consistent heuristics, then is considered more accurate if:

## IDA\*

Same as **A\***, but the idea is to reduce memory requirement of **A\*** by applying cutoff on values of **f(N)**.

* ***Pros***
  + Still complete and Optimal
  + Requires less memory than **A\***
  + Avoids the overhead to sort fringe
* ***Con***
  + Can’t avoid revisiting states not on the current path
  + Available memory is poorly used