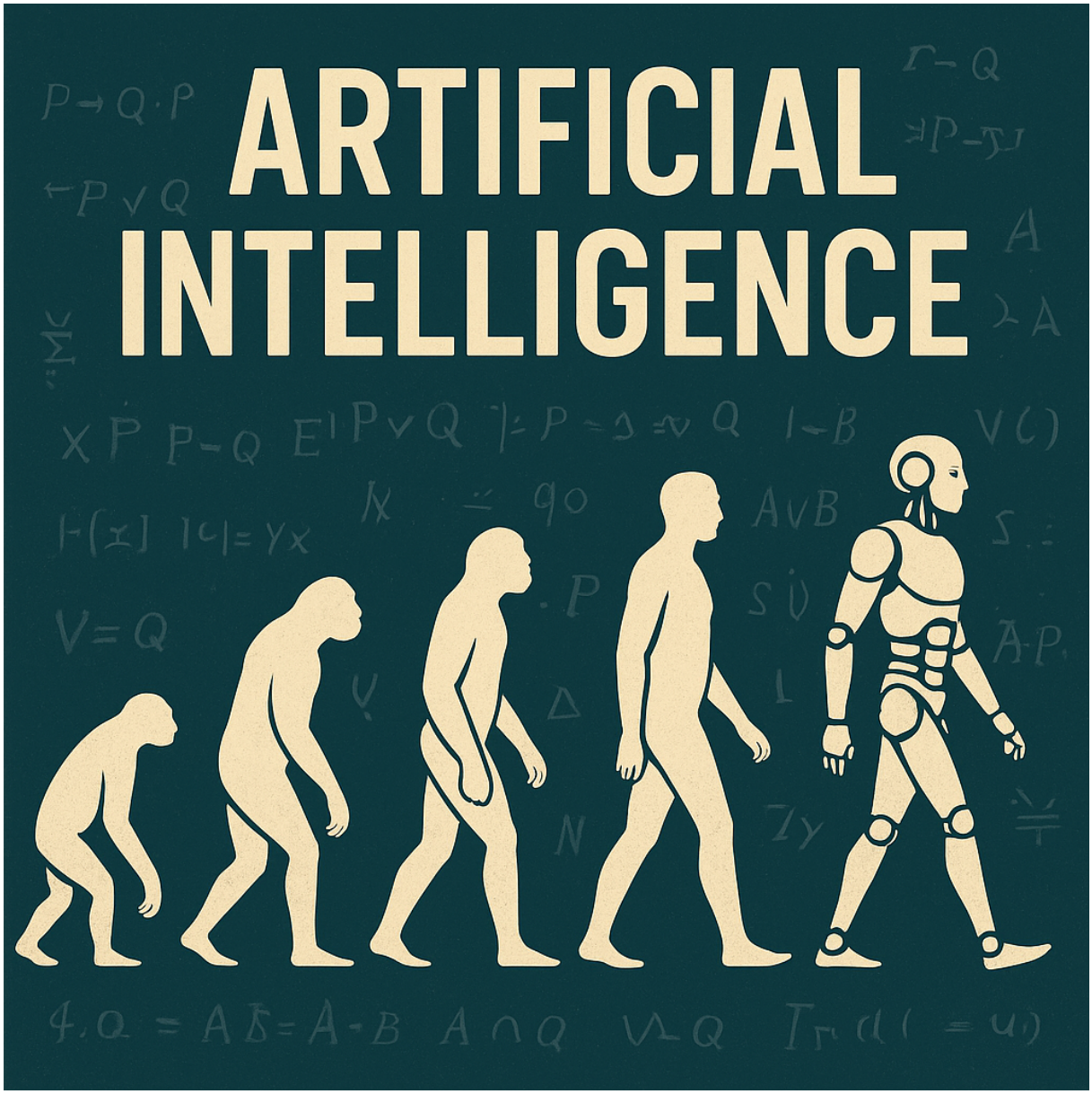
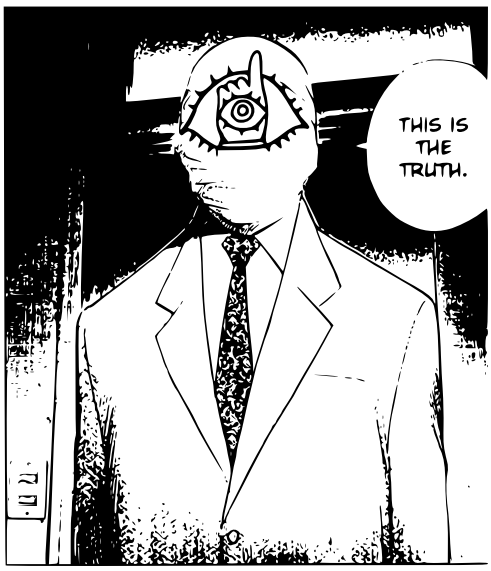
Marco Casu



SAPIENZA
UNIVERSITÀ DI ROMA

Faculty of Information Engineering, Computer Science and Statistics
Department of Computer, Control and Management Engineering
Master's degree in Artificial Intelligence and Robotics

This document summarizes and presents the topics for the Artificial intelligence course for the Master's degree in Artificial Intelligence and Robotics at Sapienza University of Rome. The document is free for any use. If the reader notices any typos, they are kindly requested to report them to the author.



CONTENTS

CHAPTER

1

INTRODUZIONE

1.1 Basic Definitions

In the context of the artificial intelligence, an **agent** is an entity that can

- Perceive the environment through *sensors* (percepts)
- Act upon the environment through *actuators* (actions).

We say that an agent is **rational** if he selects the action that maximize a given *performance measure*, informally, he attempts to do "the right thing". The best case is hypothetical and often unattainable, because the agent usually can't perform all the actions needed, and can't perceive all the information about the environment.

An agent has a performance measure M and a set A of all possible actions, given percept a sequence A and knowledge K (data), he has to select the next action $a \in A$, is a map

$$M \times P \times K \longrightarrow A. \tag{1.1}$$

An action a is optimal if it maximize the expected value of M, given the sequence P and the knowledge K. An agent is rational if he always chooses the optimal action. More specifically, an agent consists in two components:

- an architecture which provides an interface to the environment
- a program executed on that architecture.

There are some limitation that we aren't considering, such as the fact that determining the optimal choice could take too much time or memory on the architecture.

1.1.1 Types of Agents

There are different kinds of agents, a **Table Driven Agent** is the simplest form of agent architecture. It's essentially a look-up table that maps every possible sequence of percepts (what the agent has sensed so far) to a corresponding action the agent should take. His behavior can be resumed in the algorithm 1.

A **Reflex Agent** consists in three components:

- sensors to get information from the environment

Algorithm 1 Table Driven Agent

Require: *percepts*
persistent: *percepts*, a sequence, initially empty
table, a table of actions, indexed by percept sequences, initially fully specified
append *percept* to the end of *percepts*
action←`LookUp`(*percepts,table*)
return *action*

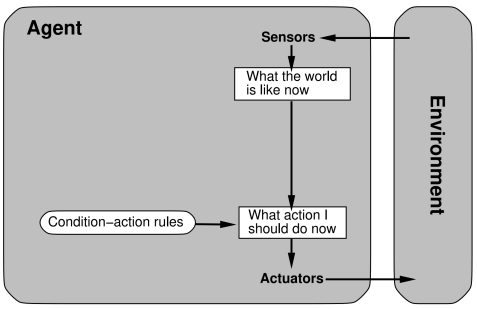


Figure 1.1: Reflex agent diagram

- a decision making process, in form of a *condition-action rules*, typically looks like `IF (condition) THEN (action)`.
- actuators, the outputs that allow the agent to affect or change the environment.

A **Model-Based Reflex Agent** is an enhanced version of the previous one, the key enhancement here is the inclusion of an *Internal State* and a *Model of the World* to make up for the agent's limited view of the environment. The internal state cannot simply be the last thing the agent saw; it needs to be updated to reflect reality. This is done using a Model of the World, which contains two key pieces of knowledge:

- How the world evolves independently of the agent, his accounts for changes in the environment that occur regardless of the agent's actions (e.g., a clock ticking, an external event).
- How the agent's own actions affect the world, this is the effect of the agent's previous action (e.g., if the agent drove forward, its position changed).

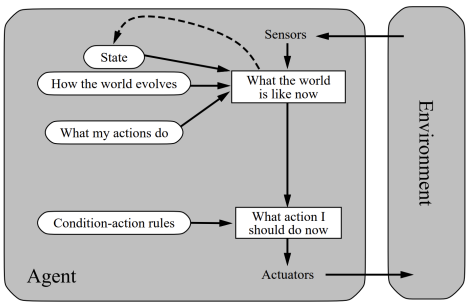


Figure 1.2: Model Based Reflex agent diagram

If a model based reflex agent consider the future prospective, is a **Goal Based Agent**, as shown in figure 1.3.

A **Utility Based Agent** is equipped with a *utility function* that maps a state to a number which represents how desirable the state is. Agent's utility function is an internalization of the performance function.

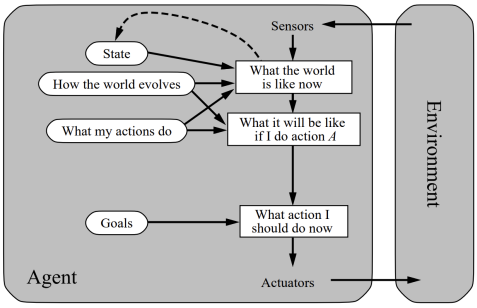


Figure 1.3: Goal Based agent diagram

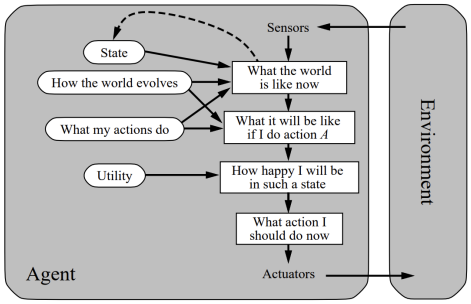


Figure 1.4: Utility Based agent diagram

A **Learning Agent** is an architecture designed to improve its efficiency over time by separating four functions:

- the performance element selects actions
- the critic provides feedback on those actions against a standard
- the learning element uses this feedback to update the agent's internal knowledge
- the problem generator suggests exploratory actions to gain new knowledge. This structure enables the agent to continuously adapt and improve its decision-making.

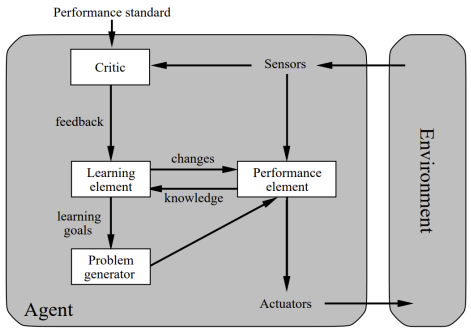


Figure 1.5: Learning agent diagram

An agent can be classified in one of the following groups:

- a **domain specific agent** is a solver specific to a particular problem (such as playing chess), is usually more efficient.
- a **general agent** is a solver for general problems, such as learning the rule of any board game, is usually more intelligent but less efficient.

1.1.2 The Environment

An environment can be classified in terms of different attributes:

- An environment can be **fully observable** if all the relevant information are accessible to the sensors, otherwise is **partially observable**.
- If there are no uncertainty, the environment is **deterministic**. An environment is **stochastic** if uncertainty is quantified by using probabilities, otherwise is **non deterministic** if uncertainty is managed as actions with multiple outcomes.
- An environment is **episodic** if the correctness of an action can be evaluated instantly, otherwise if are evaluated in the future developments, is **sequential**.
- An environment can be **static** or **dynamic**, if itdoes not change, but the agent's performance score changes, the environment is called **semi-dynamic**.
- An environment can be perceived as **discrete** or **continuous**.
- In a single environment there may be multiple agent, that can be **competitive** or **cooperative**.

Many sub-areas of AI can be classified by:

- Domain-specific vs. general.
- The environment.
- Particular agent architectures sometimes also play a role, especially in Robotics.

It follows a classification of some areas in terms of the attributes we discussed:

- **Classical Search**, the environment is
  - fully observable
  - deterministic
  - static
  - sequential
  - discrete
  - single-agent

  and the approach is *domain specific*.
- **Planning**, the environment is
  - fully observable
  - deterministic
  - static
  - sequential
  - discrete
  - single-agent

  and the approach is *general*.
- **Adversarial Search**, the environment is
  - fully observable
  - deterministic
  - static
  - sequential
  - discrete
  - multi-agent

and the approach is *domain specific*.

- **General Game Playing**, the environment is
  - fully observable
  - deterministic
  - static
  - sequential
  - discrete
  - multi-agent

  and the approach is *general*.
- **Constraint Satisfaction & Reasoning**, the environment is
  - fully observable
  - deterministic
  - static
  - episodic
  - discrete
  - single-agent

  and the approach is *general*.
- **Probabilistic Reasoning**, the environment is
  - partially observable
  - stochastic
  - static
  - episodic
  - discrete
  - single-agent

  and the approach is *general*.

CHAPTER

2

SEARCH PROBLEMS

2.1 Classical Search

Let's consider two basic example of classical search problems, the first one is the following:

Starting from Zurigo, we would like to find a route to Zagabria. We have an initial state (Zurigo), and we have to apply actions (drive) to reach the goal state (Zagabria). Another example is the following, we want to solve the tiles-puzzle game, shown in figure 2.1, to reach the left state, starting from the right one, the actions to perform is the move of the tiles. A performance measure could be to minimize the summed-up action costs.

9	2	12	6
5	7	14	13
3	4	1	11
15	10	8	

—

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

Figure 2.1: The tile game

In the classical search context, we restrict the agent's environment to a very simple setting, with a finite number of states and actions, a single agent, a fully observable stati environment that doesn't evolve, given that assumption, the classical search problems are the simplest one, despite that, are very important problems in practice.

Every problem specifies a state space.

Definition 1 *A **State Space** is a 6-tuple $\Theta = (S, A, c, T, I, S^G)$ where:*

- *S is a finite set of the states.*
- *A is a finite set of actions.*
- *$c : A \to \mathbb{R}^+$ is the cost function.*
- *$T \subseteq S \times A \times S$ is the transition relation, that describes how an action on a given state make the agent evolve to the next state. We assume that the problem is deterministic, so for all $s \in S$, $a \in A$, if $(s, a, s') \in T$ and $(s, a, s'') \in T$ then $s' = s''$.*
- *$I \in S$ is the initial state*
- *$S^G \subseteq S$ is the set of the goal states, where we want to end.*

A transition (s, a, s') can be denoted $s \xrightarrow{a} s'$, we say that $s \to s'$ if $\exists a$ such that $(s, a, s') \in T$. We say that Θ has *unit costs* if $\forall a \in A, \quad c(a) = 1$. A state space can be illustrated as a directed labeled graph.

Definition 2 *Let $\Theta = (S, A, c, T, I, S^G)$ to be a state space, we say that*

- *s' is a **successor** of s if $s \to s'$*
- *s' is a **predecessor** of s if $s' \to sT$*
- *we say that s' is **reachable from** s if*

$$\exists(a_1 \dots, a_n) \subseteq A \tag{2.1}$$
$$\exists(s_2 \dots, s_{n-1}) \subseteq S \tag{2.2}$$
$$(s, a_1, s_2) \in T \tag{2.3}$$
$$(s_2, a_2, s_3) \in T \tag{2.4}$$
$$\vdots \tag{2.5}$$
$$(s_{n-1}, a_n, s') \in T \tag{2.6}$$

  we can write the sequence as follows

$$s \xrightarrow{a_1} s_2, \dots, s_{n-1} \xrightarrow{a_n} s'. \tag{2.7}$$

- *We say that s is **reachable** (without reference state) if is reachable from I.*
- *s is **solvable** if there exists $s' \in S^G$ such that s' is reachable from s, otherwise s is **dead end**.*

Definition 3 *Let $\Theta = (S, A, c, T, I, S^G)$ to be a state space, and let $s \in S$. A **solution** for s is a path from s to some goal state $s' \in S^G$. The solution is **optimal** if it's cost is minimal, let H to be the set of all possible solution (sequence of states) for s*

$$H = \{\text{paths from } s \text{ to } s' \in S^G\} = \{(s_{i0}, s_{i1}, s_{i1}, \dots s_{in}) : s_{in} \in S^G, \ s_{i0} = s\} \tag{2.8}$$

where n^i is the length of the i-th solution. The optimal solution is

$$\arg \min_{(s, s_{i1}, \dots s_{in}) \in H} \sum_{j=0}^{n^i} c(s_{ij}). \tag{2.9}$$

A solution for I is called **solution for** Θ, if such that solution exists, Θ is **solvable**.