Einführung in die KI

Niclas Kusenbach

LaTeX version: SCHOUTER

Table of Contents

1 Introduction

1.1 What is AI?

Literature:

- Empfohlenes Begleitbuch: Russel and Norvig, Artificial Intelligence: A Modern Approach, 4. Edition 2020.

1.1.1 Definitionen (Definitions)

1.1.2 Definitions

There is no easy, official definition for AI. Two classic definitions are:

- **John McCarthy (1971):** "The science and engineering of making intelligent machines, especially intelligent computer programs." AI does not have to confine itself to methods that are biologically observable.
- **Marvin Minsky (1969):** "The science of making machines do things that would require intelligence if done by men".

1.1.3 Categories of AI

AI definitions can be classified along two dimensions

1. Thought processes/reasoning vs. behavior/action
2. Success according to human standards vs. success according to an ideal concept of intelligence (rationality)

- **Systems that think like humans:**
  - Cognitive Science.
  - Builds on cognitive models validated by psychological experiments and neurological data.
- **Systems that act like humans:**
  - The **Turing Test**
- **Systems that think rationally:**
  - Focus on "Laws of Thoughts," correct argument processes.
- **Systems that act rationally:**
  - Focus on "doing the right thing" (**Rational Behavior**).
  - A rationally acting system maximizes the achievement of its goals based on the available information.
  - This is more general than rational thinking (as a provably correct action often does not exist) and more amenable to analysis.

1.1.4 General vs. Narrow AI

- **General (Strong) AI:** Can handle *any* intellectual task that a human can. This is a research goal.
- **Narrow (Weak) AI:** Is specified to deal with a *concrete* or a set of specified tasks. This is what we currently use primarily.

1.2 What is Intelligence?

1.2.1 The Turing Test

- **Question:** When does a system behave intelligently?

- **Assumption:** An entity is intelligent if it cannot be distinguished from another intelligent entity by observing its behavior.
- **Test:** A human interrogator interacts ”blind” (e.g., via text) with two players (A and B), one of whom is a human and one a computer.
- **Goal:** If the interrogator cannot determine which player... is a computer... the computer is said to pass the test.
- **Relevance:** The test is still relevant, requires major components of AI (knowledge, reasoning, language, learning), but is hard/not reproducible and not amenable to mathematical analysis.

1.2.2 The Chinese Room Argument

- **Question:** Is intelligence the same as intelligent behavior?
- **Assumption:** Even if a machine behaves in an intelligent manner, it does not have to be intelligent at all (i.e., without understanding).
- **Thought Experiment:** A person who doesn't know Chinese is locked in a room. They receive Chinese notes (questions) and have a detailed instruction book telling them which Chinese symbols (answers) to output based on the input symbols, without understanding it at all.
- **Result:** From the outside, the room ”understands” Chinese (it behaves intelligently), but the person inside understands nothing.
- **Follow-up Question:** Is a self-driving car intelligent?

1.3 Foundations, Taxonomy & Limits

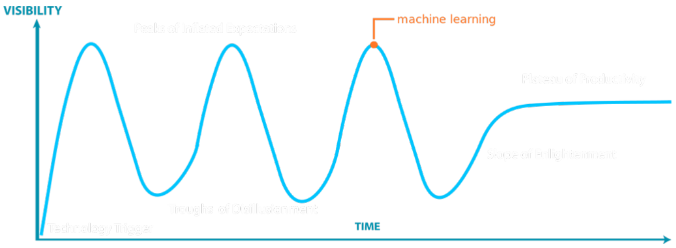
1.3.1 Foundations of AI

AI is an interdisciplinary field built on contributions from many areas:

- **Philosophy:** Logic, reasoning, rationality, mind as a physical system.
- **Mathematics:** Formal representation and proof, computation, probability.
- **Psychology:** adaptation, phenomena of perception and motor control.
- **Economics:** formal theory of rational decisions, game theory.
- **Linguistics:** knowledge representation, grammar.
- **Neuroscience:** physical substrate for mental activities.
- **Control theory:** ...optimal agent design.

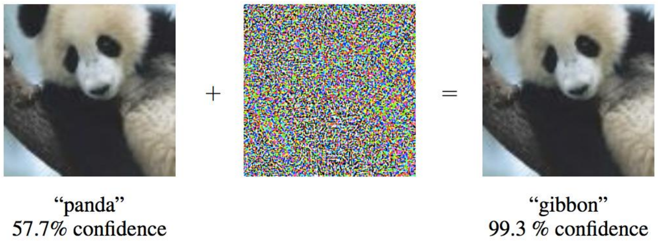
1.3.2 Taxonomy and History

- **Taxonomy: Artificial Intelligence** is the broadest field. **Machine Learning (ML)** is a subfield of AI. **Deep Learning** is a subfield of ML.
- **Subdisciplines of AI:** Include Machine Learning, Deep Learning, Search and Optimization, Robotics, Natural Language Processing (NLP), Computer Vision (CV), and Cognitive Science.
- **History:** The development of AI occurred in cycles, often called ”AI Winters”. Hype phases (”Peaks of Inflated Expectations”) existed for ”neural networks”, ”expert systems”, and ”machine learning”.



1.3.3 Limits of Current AI

- **"A.I. is harder than you think":**
  - Current AI is often isolated to single problems.
  - AI models are **not without bias**.
  - There are **fundamental differences** in how AI perceives the world/environment.
- **AI can be tricked (Adversarial Examples):**
  - AI systems can be manipulated by perturbations (noise) often invisible to humans.
  - Example: An image of a "panda" is classified as a "gibbon" with high confidence after adding noise.



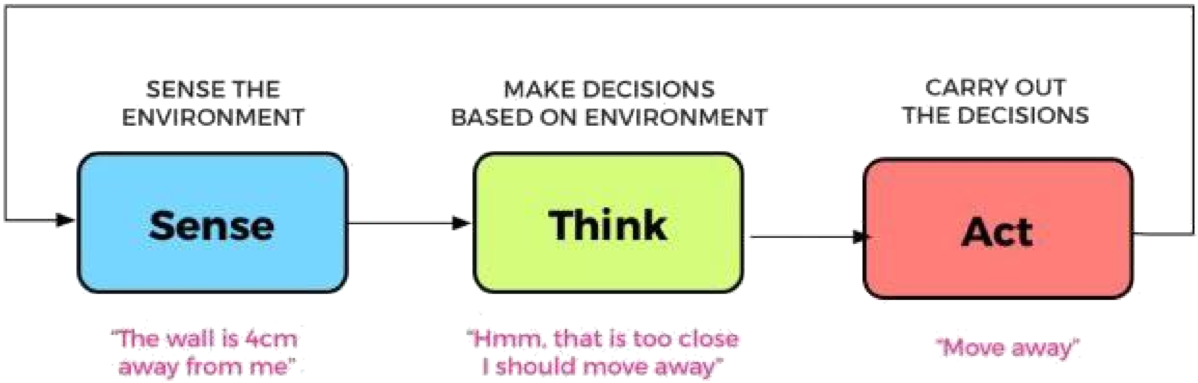
2 AI Systems: Agents and Environments

Definition: AI System

An AI system is defined as the study of (rational) **agents** and their **environments**. The system has two main parts:

1. **Agent:** Anything that can be viewed as perceiving its environment through **sensors** and acting upon that environment through **actuators**.
2. **Environment:** The surroundings or conditions in which the agent lives or operates. This can be real (e.g., streets for a self-driving car) or artificial (e.g., a chessboard).

The agent follows a continuous **Sense** → **Think** → **Act** loop.


SENSE THE ENVIRONMENT
MAKE DECISIONS BASED ON ENVIRONMENT
CARRY OUT THE DECISIONS
Sense
Think
Act
"The wall is 4cm away from me"
"Hmm, that is too close I should move away"
"Move away"

2.1 Rationality

Rationality

- A **rational agent** is one that ”does the right thing”.
- A **rational action** is one that maximizes the agent’s performance and yields the best positive outcome.
- **Key Point:** Rationality maximizes **expected** performance, not necessarily the *optimal* outcome. E.g., not playing the lottery is rational (positive expected outcome), even if playing could lead to the optimal outcome (winning).
- Rationality is **not** omniscient. An omniscient agent would know the *actual* outcome of its actions, which is impossible in reality.

A **performance measure** is a function that evaluates a sequence of actions.

General Rule for Design

Design the performance measure based on the **desired outcome**, not the desired agent behaviour.

2.2 Characteristics of Environments

The design of an agent heavily depends on the type of environment it operates in. Environments are characterized along several key dimensions.

Environment Dimensions

- **Discrete vs. Continuous:** Does the environment have a limited, countable number of distinct states (e.g., chess) or is it continuous (e.g., position and speed of a self-driving car)?
- **Observable vs. Partially/Unobservable:** Can the agent's sensors determine the *complete* state of the environment at each time point? If not, it is **partially observable** (e.g., a taxi cannot know pedestrian intentions, poker agent cannot see opponent's cards).
- **Static vs. Dynamic:** Does the environment change while the agent is acting/deliberating? A crossword puzzle is **static**; taxi driving is **dynamic** (other cars move).
- **Single Agent vs. Multiple Agents:** Is the agent operating by itself? Or does the environment contain other agents (e.g., other drivers, poker players)?
- **Accessible vs. Inaccessible:** Can the agent obtain *complete and accurate* information about the environment's state?
- **Deterministic vs. Non-deterministic (Stochastic):** Is the next state of the environment completely determined by the current state and the agent's action? Chess is **deterministic**. A self-driving car is **non-deterministic** (turning the wheel can have slightly different effects due to road friction, wind, etc.).
- **Episodic vs. Sequential:** In an **episodic** environment, the agent's experience is divided into ”episodes”. The quality of its action depends only on the current episode (perceive → act). In a **sequential** environment, the agent requires memory of past actions to make the best decision.

Key Distinction: Observable vs. Accessible

- **Accessibility** concerns the environment itself: whether the information exists and can *in principle* be obtained.
- **Observability** concerns the agent's *sensors*: whether they can actually perceive that information.

Environment	Discrete?	Observable?	Static?	Single Agent?	Accessible?	Deterministic?	Episodic?
Chess	Discrete	Observable	Static	Multi-Agent	Accessible	Deterministic	Sequential
Solitaire	Discrete	Observable	Static	Single Agent	Accessible	Deterministic	Sequential
Poker	Discrete	Partially Observable	Static	Multi-Agent	Partially Accessible	Stochastic	Sequential
Self-Driving	Continuous	Partially Observable	Dynamic	Single Agent	Inaccessible	Stochastic	Sequential
Medical Diagnosis	Discrete	Partially Observable	Static	Single Agent	Inaccessible	Stochastic	Episodic

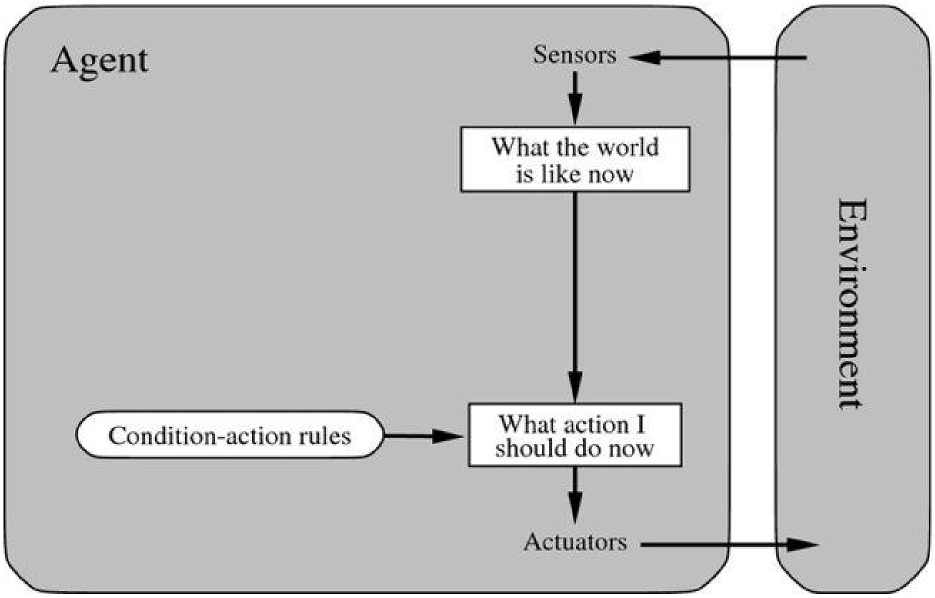
Characteristics of various environments

2.3 Types of Agents

Agents are categorized based on their perceived intelligence and complexity.

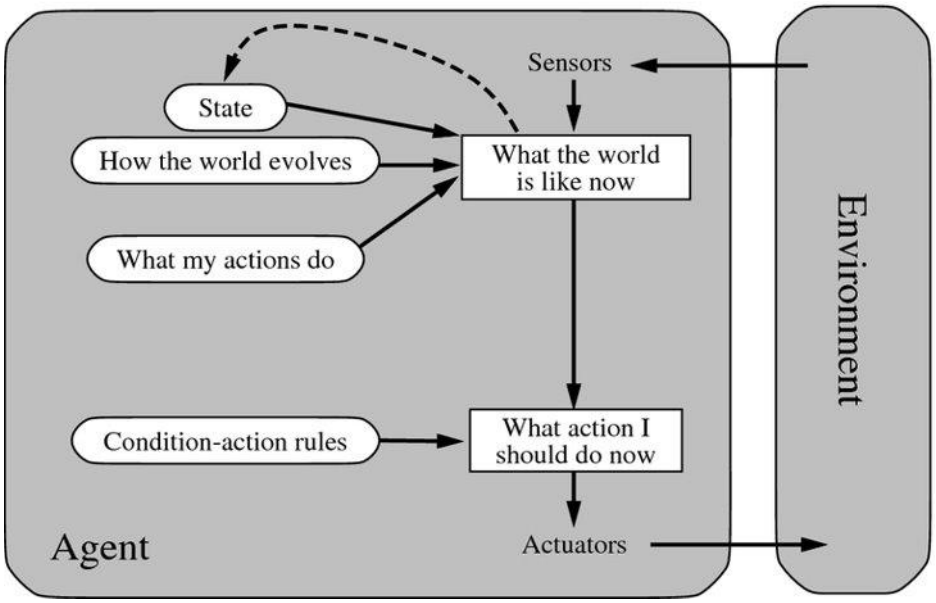
2.3.1 Reflex Agent

- Selects actions based **only on the current percept**, ignoring the percept history.
- Implemented with simple **condition-action rules**.
- Example: A thermostat (IF temp $<$ 20°C → turn on heater).
- **Problem:** Very limited. No knowledge of anything it cannot actively perceive.



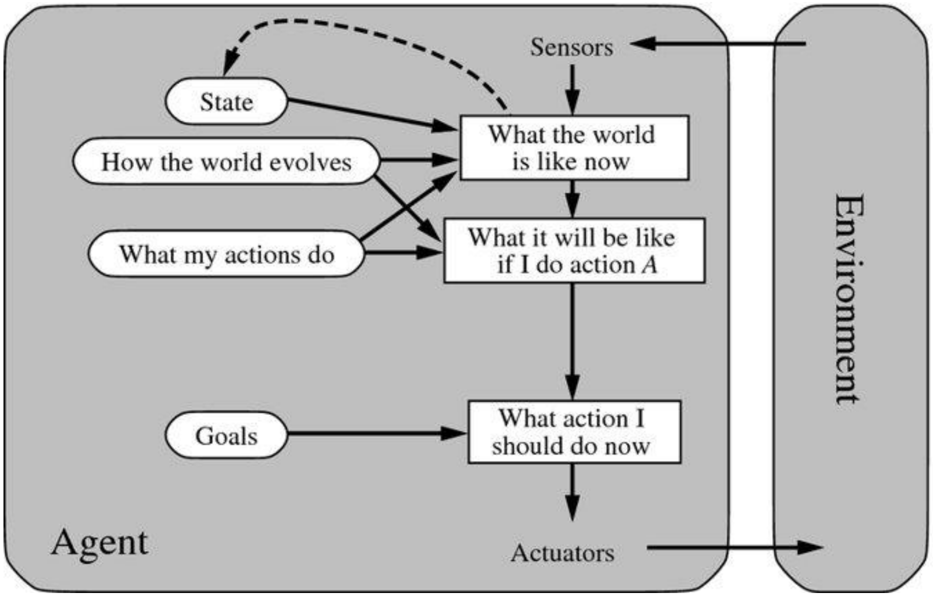
2.3.2 Model-based Agent

- These agents **keep track of the world state**.
- They maintain an **internal state (a world model)** that describes how the world evolves and how the agent's actions affect it.
- This allows the agent to handle partially observable environments.
- Example: A warehouse robot tracking inventory positions.



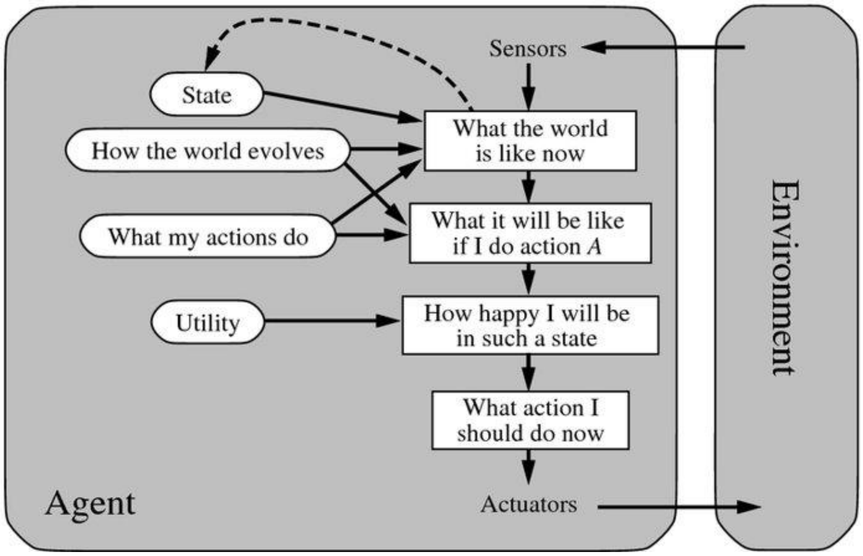
2.3.3 Goal-based Agent

- Builds on a model-based agent, but also knows what states are **desirable** (i.e., it has **goals**).
- This allows the agent to make decisions by considering the future, asking "What will happen if I do action A?" and "Will that action achieve my goal?".
- Example: A chess agent whose goal is to checkmate the opponent.


Sensors
State
How the world evolves
What my actions do
What the world is like now
What it will be like if I do action A
Goals
What action I should do now
Agent
Actuators
Environment

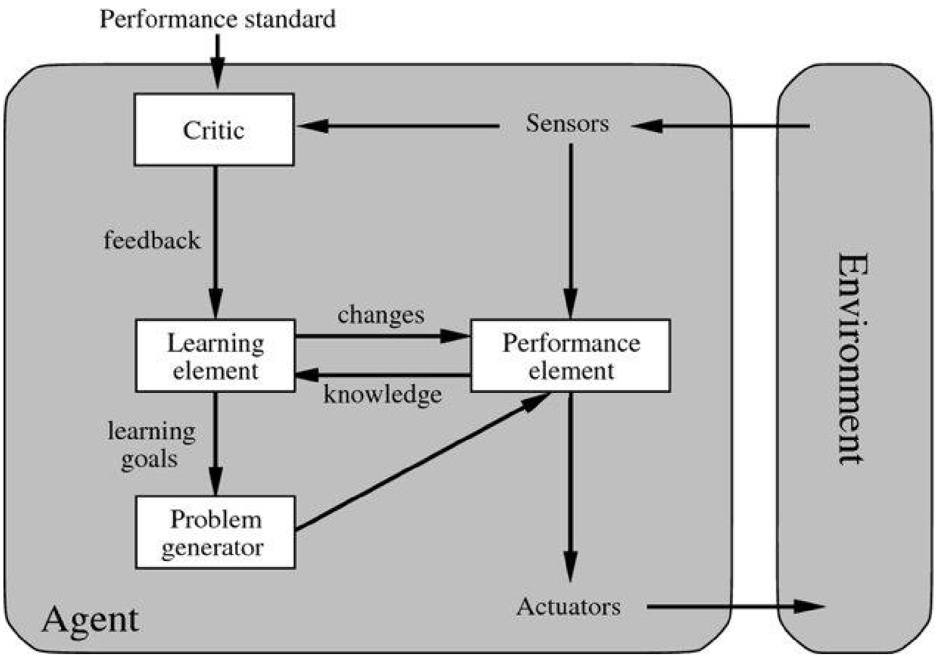
2.3.4 Utility-based Agent

- Goals provide a binary distinction (achieved / not-achieved). A **utility function** provides a continuous scale, rating each state based on the desired result (”how happy” the agent is).
- This is crucial for resolving **conflicting goals** (e.g., is speed or safety more important for a self-driving car?).
- Allows the agent to choose the action that maximizes its **expected utility**.


Sensors
State
How the world evolves
What my actions do
What the world is like now
What it will be like if I do action A
Utility
How happy I will be in such a state
What action I should do now
Agent
Actuators
Environment

2.3.5 Learning Agent

- Employs a **learning element** to gradually improve and become more knowledgeable over time.
- Can learn from its past experiences and adapt automatically.
- More robust in unknown environments.



Four Components of a Learning Agent

1. **Learning Element:** Responsible for making improvements by learning from the environment.
2. **Critic:** Gives feedback on how well the agent is doing with respect to a fixed performance standard.
3. **Performance Element:** Responsible for selecting actions (this is the ”agent” part).
4. **Problem Generator:** Responsible for suggesting actions that will lead to new (and potentially informative) experiences.

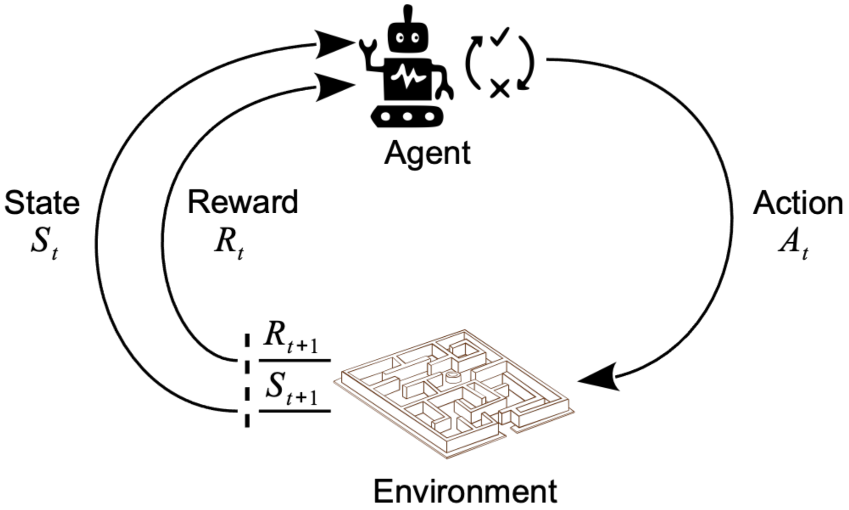
Agent Types Summary

- **Reflex agent:** reacts.
- **Model-based agent:** remembers.
- **Goal-based agent:** plans.
- **Utility-based agent:** optimizes.
- **Learning agent:** improves itself over time.

2.4 How to Make Agents Intelligent

There are several high-level approaches to selecting intelligent actions:

- **Search Algorithms:** Understand ”finding a good action” as a search problem and use tree-based algorithms to find a solution (path to a goal).
- **Reinforcement Learning (RL):** Based on trial and error, similar to animal conditioning. The agent receives **rewards** (positive) or **pain/punishments** (negative) from the environment and learns to choose actions that maximize its cumulative reward.



- **Genetic Algorithms (GAs):** Inspired by Darwinian evolution (”survival of the fittest”). A **population** of agents is generated, evaluated by a **performance function**, and the best ones are ”bred” (using **crossover** and **mutation**) to create a new, potentially better, generation.

3 Uninformed and Informed Search

3.1 Problem Formulation

Problem-solving agents are result-driven. They always focus on satisfying their goals, i.e., solving the problem. While problems are often given in a human-understandable way, we need to reformulate the problem for our agent. These agents employ algorithms to find solutions.

Steps to formulate a solvable problem:

1. **Formulate the goal**
2. **Formulate the problem** given the goal

3.1.1 Key Terminology

The State Space / States

A state describes a possible situation in our environment. The **state space** is a set of all possible situations (states).

Transition / Action

Transitions describe possible actions to take between one state and another. We only count direct transitions between two states (single actions).

Costs

Often transitions aren't alike and differ. We express this by adding a "cost" to each action. Often the goal in search algorithms is to **minimize the cost** to reach the goal.

A **single state problem** is defined by 4 items:

1. **State space and Initial state** Description of all possible states and the initial environment as state.
2. **Description of actions** Typically a function that maps a state to a set of possible actions in this state.
3. **Goal test** Typically a function to test if the current state fulfills the goal definition.
4. **Costs** A cost function that maps actions to its cost. An easy way is to have additive costs (sum of costs for all actions taken).

3.1.2 The State-Space Graph

The state space is the set of all states reachable from the initial state. It is implicitly defined by the initial state and the successor function, forming a **state-space graph**.

- **Path:** A sequence of states connected by a sequence of actions.
- **Solution:** A path that leads from the initial state to a goal state.
- **Optimal Solution:** A solution with the minimum path cost.

3.1.3 Core Search Definitions

Planning Problem

A planning problem is one in which we have an initial state and want to transform it into a desired goal, considering future actions and their outcomes.

Search

The process of finding the (optimal) solution for such a problem in the form of a sequence of actions.

3.2 Search Fundamentals

3.2.1 Tree Search vs. Graph Search

The state-space graph can be explored by building a **search tree**.

- **Tree Search:** Treats the state space as a tree. It does not keep track of visited states, so it might re-explore the same state via a different path. This can lead to exponential work for problems with loops or redundant paths.
- **Graph Search:** Remembers states that have been visited in an **explored set** (or ”closed set”). It avoids expanding states that are already in the explored set, thus handling loops and redundant paths efficiently.

3.2.2 States vs. Nodes

State

Representation of a physical configuration. Describes a specific situation in our environment (e.g., ”in Arad”).

Node

A data structure to represent a part of a search tree. It includes a **state**, a **parent node**, the **action** taken, the **path cost** ($g(n)$), and the **depth** (e.g., ”the path Arad → Sibiu”).

3.2.3 Key Search Tree Terminology

Fringe

The set of all nodes at the end of all visited paths is called the fringe. (Also known as **frontier** or ”open set”). These are the nodes available for expansion.

Depth

Number of levels in the search tree.

3.2.4 Evaluating Search Strategies

Search strategies are evaluated along the following dimensions:

- **Completeness:** Does it always find a solution if one exists?
- **Time Complexity:** Number of node expansions.
- **Space Complexity:** Maximum number of nodes in memory.
- **Optimality:** Does it always find the optimal (least-cost) solution?

Complexity is measured in terms of:

- b: maximum **branching factor** of the search tree.
- d: the **depth** of the optimal solution.
- m: the **maximum depth** of the state space (may be ∞).

3.3 Uninformed Search Strategies

Uninformed Search

Do not have any information about the problem except the problem definition. (Also called **Blind Search**).

Breadth-First Search (BFS)

A special case of Uniform-Cost Search where all step costs are equal. It starts at the tree root and explores the tree **level by level**. It uses a FIFO (First-In-First-Out) queue for the fringe.

- **Completeness:** Yes.
- **Time:** $O(b^d)$ (The summary table uses $O(b^{d+1})$).
- **Space:** $O(b^d)$. Memory consumption is its biggest drawback.
- **Optimality:** Yes (if all costs are equal).

Uniform-Cost Search (UCS)

Each node is associated with a cost, which accumulates over the path. UCS expands the node with the **lowest cumulative path cost** ($g(n)$). It is often implemented with a **priority queue**.

- **Completeness:** Yes (if step costs are positive, i.e., $> \epsilon > 0$).
- **Time:** $O(b^(1 + \lfloor C^/\epsilon \rfloor))$, where C is the cost of the optimal solution.
- **Space:** $O(b^(1 + \lfloor C^*/\epsilon \rfloor))$.
- **Optimality:** Yes.

Depth-First Search (DFS)

Starts at the tree root and explores as far as possible along one branch before backtracking. It uses a LIFO (Last-In-First-Out) stack for the fringe.

- **Completeness:** No. Fails in infinite-depth spaces or spaces with loops.
- **Time:** $O(b^m)$, where m is the max depth. Can be terrible if $m \gg d$.
- **Space:** $O(b \times m)$. This linear space complexity is its key advantage.
- **Optimality:** No.

Depth-limited Search (DLS)

A variation of DFS where the search is limited to a predetermined depth l. Nodes at depth l are not expanded.

- **Completeness:** No (if $l < d$).
- **Time:** $O(b^l)$.

- **Space:** $O(b \times l)$.
- **Optimality:** No.

Iterative Deepening Search (IDS)

Combines the benefits of BFS and DFS. It runs DLS repeatedly with increasing depth limits: $l = 0, 1, 2, \dots, d$.

- **Completeness:** Yes.
- **Time:** $O(b^d)$ (Despite re-generating upper levels, the overhead is small).
- **Space:** $O(b \times d)$.
- **Optimality:** Yes (if costs are uniform).

Bidirectional Search

Performs two searches simultaneously: one forward from the initial state, one backward from the goal state. Stops when the two searches meet.

- **Completeness:** Yes.
- **Time:** $O(b^{d/2})$.
- **Space:** $O(b^{d/2})$.
- **Notes:** Only possible if actions can be reversed.

3.3.1 Summary of Uninformed Strategies

Criterion	**Breadth-First**	**Uniform-Cost**	**Depth-First**	**Depth-Limited**	**Iterative Deepening**	heightComplete?
Yes	Yes	No	Yes, if $l \geq d$	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 height					

*Comparison of uninformed search strategies. (*Assumes uniform step costs or $l \geq d$ where applicable).*

3.4 Informed Search Strategies

Informed Search

Have additional knowledge about the problem (beyond the definition) and an idea of where to ”look” for solutions.

This ”hint” is provided by a heuristic function.

3.4.1 Heuristics

Heuristic $h(n)$

Informally denotes a ”rule of thumb”. In tree-search, a heuristic is a function $h(n)$ that **estimates the remaining cost** to reach the goal from node n.

3.4.2 Greedy Best-first Search

Greedy Best-first Search

Uses an evaluation function $f(n) = h(n)$ to estimate the cost from node n to the goal. It expands the node that appears to be closest to the goal, according to the heuristic.

- **Completeness:** No. Can get stuck in loops. (Complete in finite spaces with loop detection).
- **Time:** Worst case $O(b^m)$.
- **Space:** Worst case $O(b^m)$ (keeps all nodes in memory).
- **Optimality:** No. The solution depends entirely on the heuristic.

3.4.3 A* Search

A* Search

An informed tree search algorithm, building on best-first search. It is the "best-known" form. It avoids expanding paths that are already expensive.

A* evaluates nodes using the function: $f(n) = g(n) + h(n)$.

- $g(n) =$ **true cost** so far to reach node n.
- $h(n) =$ **estimated cost** to get from n to the goal.
- $f(n) =$ **estimated cost** of the cheapest solution path that goes through node n.
- **Completeness:** Yes (unless there are infinitely many nodes with $f(n) \leq f(G)$).
- **Time:** Can be exponential unless the error of the heuristic $h(n)$ is bounded.
- **Space:** Has to keep all nodes in memory. This is the primary drawback of A*.
- **Optimality:** Yes, **if the heuristic** $h(n)$ **is admissible**.

3.4.4 Heuristic Properties

Admissible Heuristics

A heuristic is **admissible** if it **never overestimates** the cost to reach a goal. Formally: $h(n) \leq h^(n)$ for all nodes n, where $h^(n)$ is the true cost from n to the goal. (e.g., straight-line distance h_{SLD} is admissible for route-finding).

Consistent Heuristics

A heuristic is **consistent** if for every node n and every successor n' generated by action a, the "triangle inequality" holds: $h(n) \leq c(n, a, n') + h(n')$. This means the heuristic difference between adjacent nodes never overestimates the actual step cost.

- **Lemma 1:** Every **consistent** heuristic is also **admissible**.
- **Lemma 2:** If $h(n)$ is consistent, then the values of $f(n)$ along any path are **non-decreasing**.

Relaxed Problems

A problem with fewer restrictions on the actions is called a relaxed problem. The cost of an optimal solution to a relaxed problem is an **admissible heuristic** for the original problem.

Example (8-puzzle):

- $h_1(n) =$ Number of misplaced tiles. (Relaxed rule: tile can move anywhere).
- $h_2(n) =$ Total Manhattan distance. (Relaxed rule: tile can move to any adjacent square).
- Both h_1 and h_2 are admissible.

Dominance

If h_1 and h_2 are both admissible and $h_2(n) \geq h_1(n)$ for all n, then h_2 **dominates** h_1.

A* will expand fewer nodes with a dominant heuristic. (e.g., for the 8-puzzle, h_2 (Manhattan) dominates h_1 (misplaced tiles)).

Combining Heuristics If we have several admissible heuristics $h_1(n), \ldots, h_m(n)$, we can combine them. $h(n) = \max h_1(n), h_2(n), \ldots, h_m(n)$ is also admissible and dominates all of its components.

3.4.5 Optimality of A

A (using Tree Search) is optimal if its heuristic $h(n)$ is admissible.

Proof (Informal):

1. Let G be an optimal goal state, with path cost C.
2. Assume for contradiction that A is about to return a suboptimal goal G_2, with path cost $g(G_2) > C$.
3. At this moment, G_2 is in the fringe. Because A chose G_2, its f-value must be the lowest, so $f(G_2) \leq f(n)$ for all other fringe nodes n.
4. Let n be any unexpanded node on a true optimal path to G. This node n must be in the fringe.
5. **Analyze $f(G_2)$:** For a goal state, $h(G_2) = 0$. So, $f(G_2) = g(G_2) + h(G_2) = g(G_2)$. Since G_2 is suboptimal, $f(G_2) = g(G_2) > C$.
6. **Analyze $f(n)$:** $f(n) = g(n) + h(n)$. Because h is admissible, $h(n) \leq h^(n)$ (where $h^(n)$ is the true cost from n to G). The true cost of the optimal path is $C^=g(n) + h^(n)$. Therefore, $f(n) = g(n) + h(n) \leq g(n) + h^(n) = C$.
7. **Contradiction:** We have shown $f(n) \leq C$ and $f(G_2) > C$. This means $f(n) < f(G_2)$. A would be forced to expand n (on the optimal path) *before* it could ever expand G_2. Thus, A* can never select a suboptimal goal. It is optimal.

3.4.6 Memory-Bounded Heuristic Search

The main problem with A* is its space complexity. Alternatives include:

1. **Iterative-deepening A* (IDA):** Like IDS, but the "depth" cutoff is the f-cost $(g + h)$.
2. **Recursive best-first search (RBFS):** Mimics best-first search using linear space by recursively re-expanding nodes and updating f-values from ancestors.
3. **(Simple) Memory-bounded A ((S)MA*):** When memory is full, drops the worst (highest f-value) leaf node.

3.5 Tree Search vs. Graph Search (Revisited)

Failure to detect repeated states can turn a linear problem into an exponential one!

Graph Search

Uses an **explored set** (or "closed set") to store all states that have been expanded. When expanding a node, its successors are only added to the fringe **if they are not in the fringe or explored set**.

Optimality of A* Graph Search

- If $h(n)$ is only **admissible**, Graph Search A* is not guaranteed to be optimal. It might find a suboptimal path to a node first, add it to the explored set, and never find the optimal path.
- If $h(n)$ is **consistent**, Graph Search A* **is optimal**.
- **Why?** A consistent heuristic guarantees that f-values are non-decreasing along any path. This means the *first* time A* expands a node n, it is *guaranteed* to have found the shortest possible path to it. Therefore, we never need to re-expand any node in the explored set.