Einführung in die KI

Niclas Kusenbach
LaTeX version: SCHOUTER

Table of Contents

1 Introduction **2**
1.1 What is AI? 2
1.1.1 Definitionen (Definitions) 2
1.1.2 Definitions 2
1.1.3 Categories of AI 2
1.1.4 General vs. Narrow AI 2
1.2 What is Intelligence? 2
1.2.1 The Turing Test 2
1.2.2 The Chinese Room Argument . . . 3
1.3 Foundations, Taxonomy & Limits 3
1.3.1 Foundations of AI 3
1.3.2 Taxonomy and History 3
1.3.3 Limits of Current AI 4

2 AI Systems: Agents and Environments **5**
2.1 Rationality 5
2.2 Characteristics of Environments 5
2.3 Types of Agents 6
2.3.1 Reflex Agent 6
2.3.2 Model-based Agent 7
2.3.3 Goal-based Agent 7
2.3.4 Utility-based Agent 8
2.3.5 Learning Agent 8
2.4 How to Make Agents Intelligent 9

3 Uninformed and Informed Search **11**
3.1 How to Solve a Problem 11
3.1.1 The State Space of a Problem . . . 11
3.2 Tree Search Algorithmus 12
3.3 Search Strategy 13
3.3.1 Uninformed Tree Search Strategies 14
3.3.2 Informed Tree Search Strategies . 16
3.3.3 Heuristics 16

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

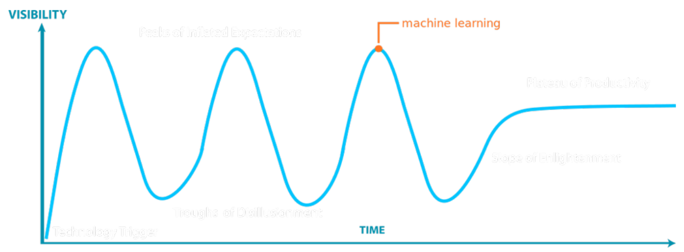
1.3.1 Foundations of AI

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.

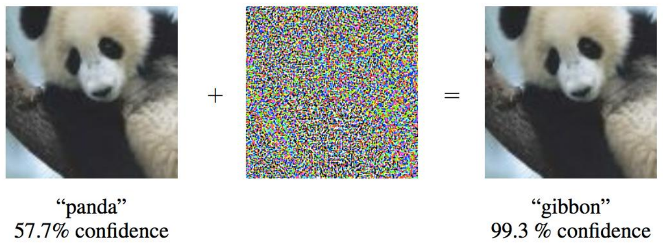
1.3.2 Taxonomy and History

- **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.



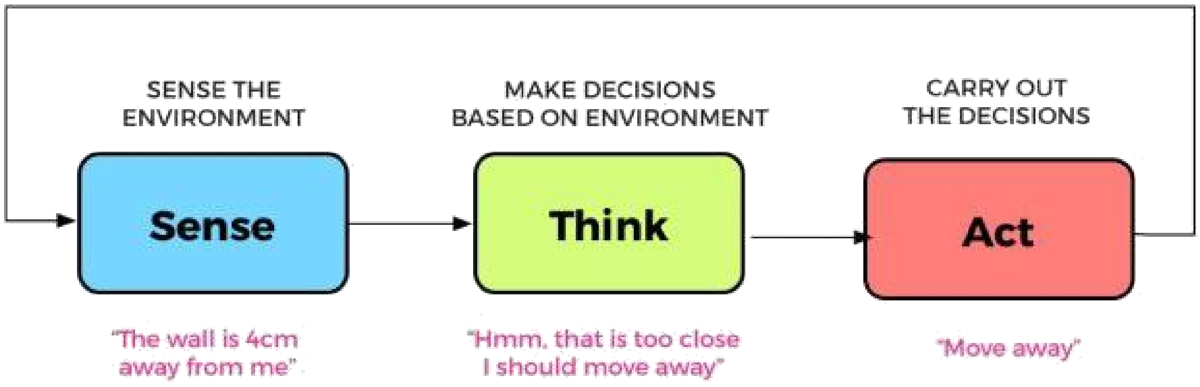
2 AI Systems: Agents and Environments

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 $\rightarrow$ 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

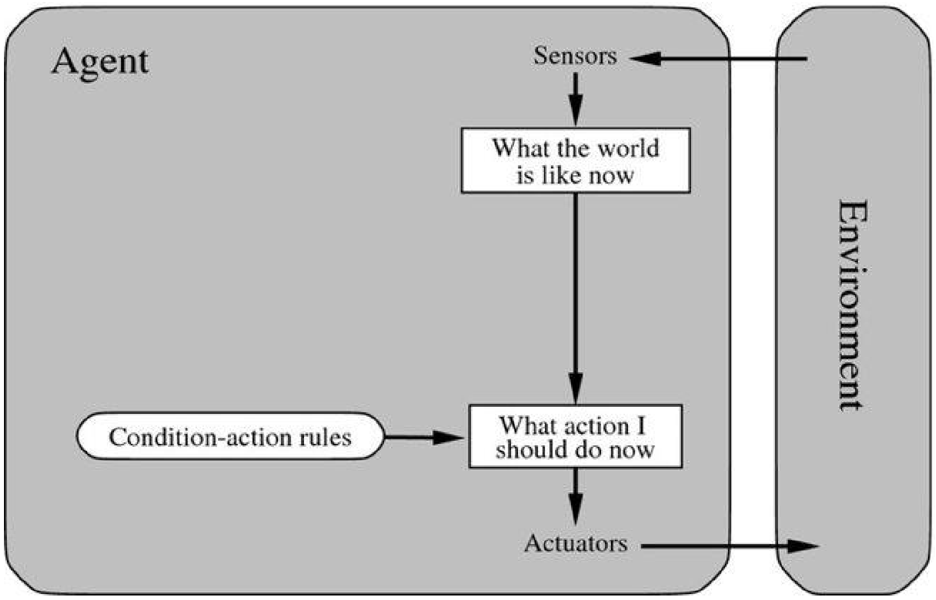
Characteristics of various environments

2.3 Types of Agents

Agents are categorized based on their perceived intelligence and complexity.

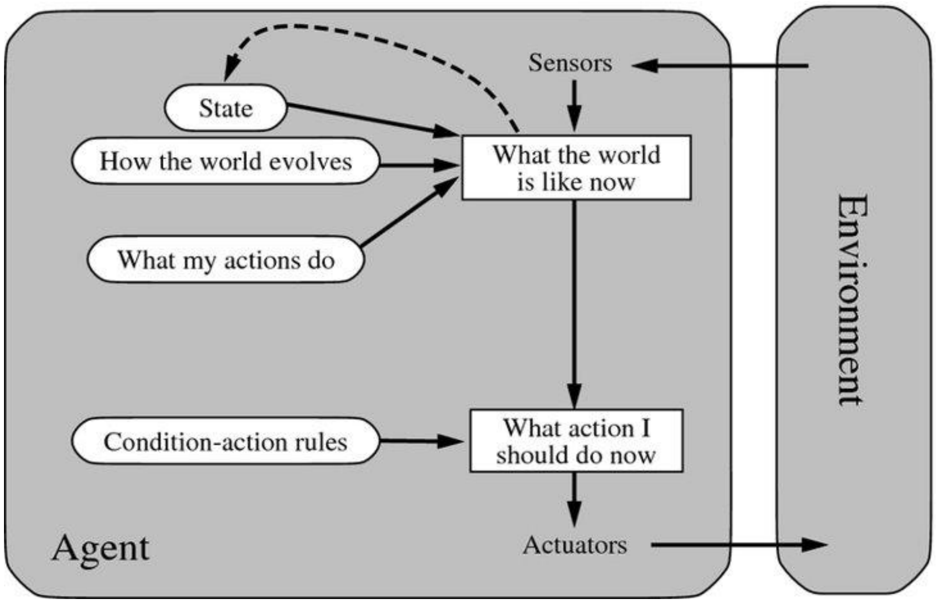
2.3.1 Reflex Agent

- 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 $\rightarrow$ turn on heater).
- **Problem:** Very limited. No knowledge of anything it cannot actively perceive.



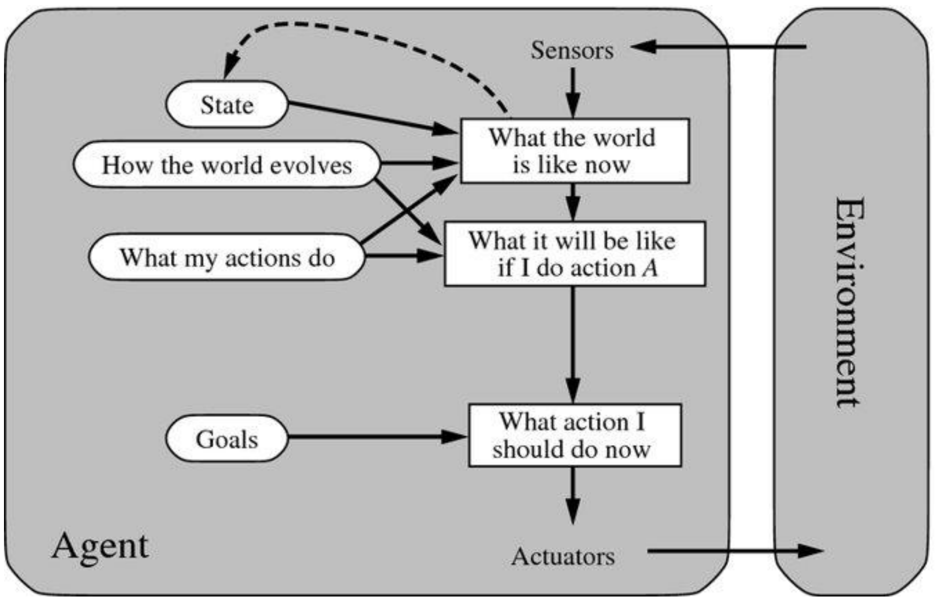
2.3.2 Model-based Agent

- 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.



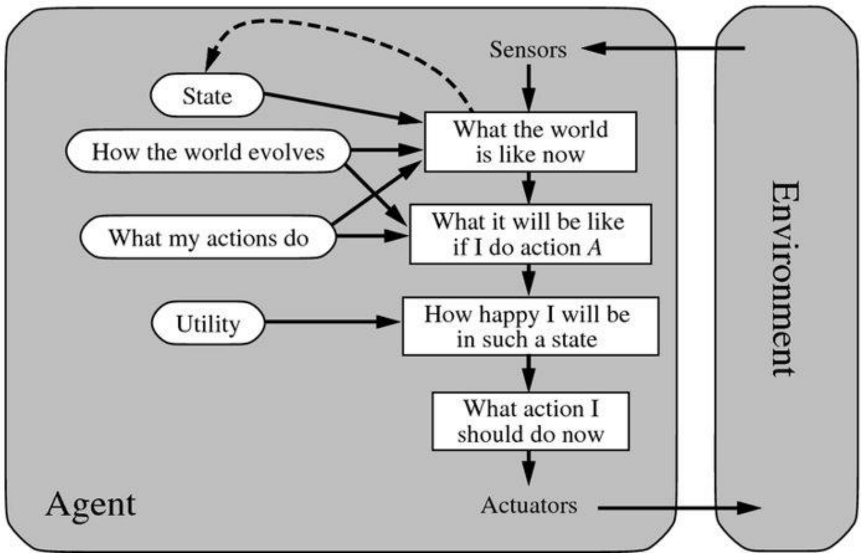
2.3.3 Goal-based Agent

- 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

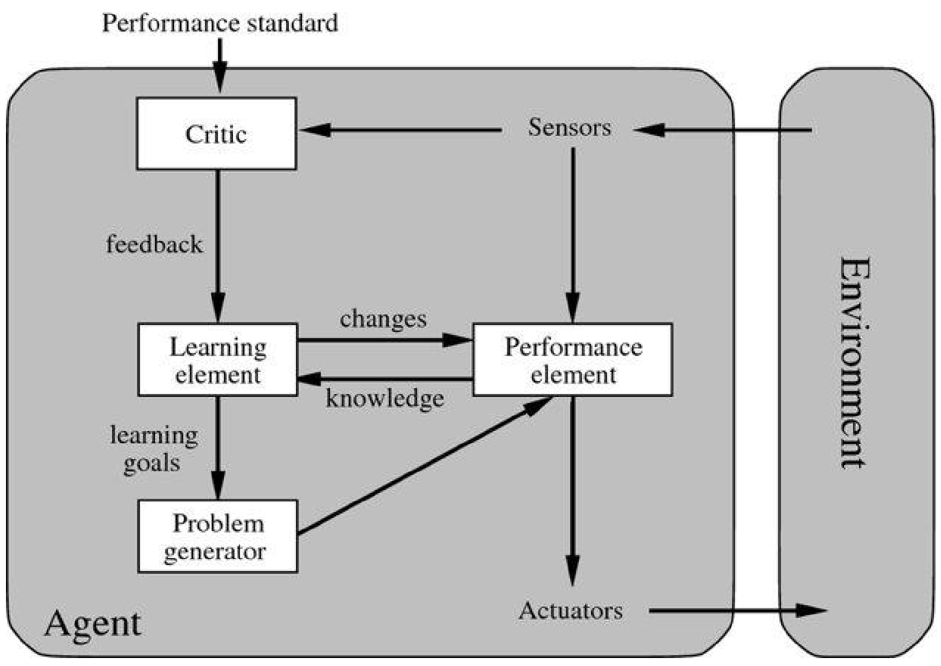
2.3.4 Utility-based Agent

- 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.

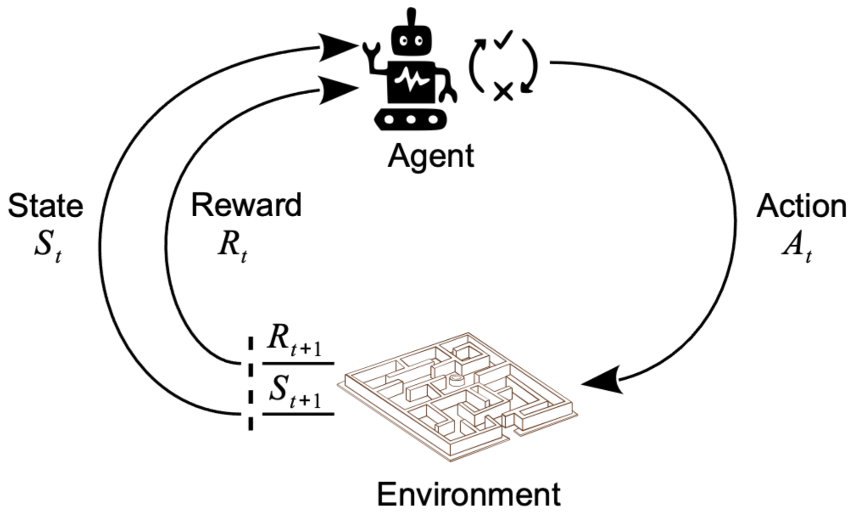
Agent Types Summary

- **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 How to Solve a Problem

Problem-solving agents are result driven. They always focus on satisfying its foals, i.e., solving the problem. While Problems are often given in a human-understandable way we need to reformulate the problem for our agent. Those agents employ algorithms to develop/find solutions.
Steps to formulate a solvable problem:

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

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 coast to reach the goal.

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 outcomes of them.

Search

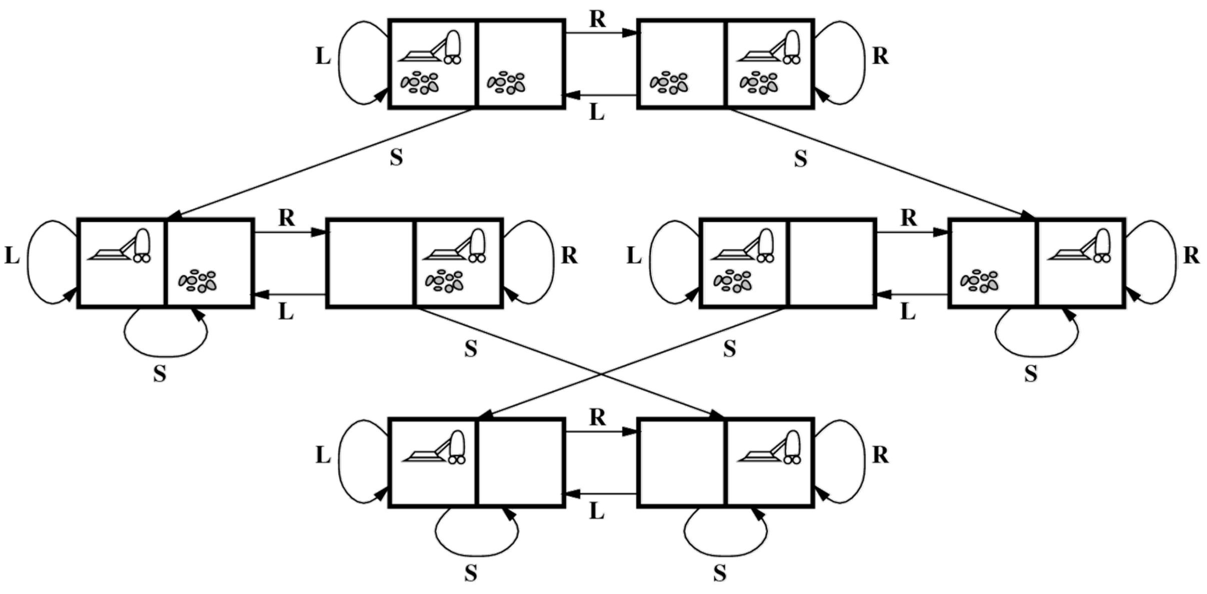
The process of finding the (optimal) solution for such a problem in form of a sequence of actions.

A single state problem is defined in 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

3.1.1 The State Space of a Problem

- The set of all states reachable form the initial state
- Implicitly defined by the initial state and the successor function, so we have a graph, the state-space graph

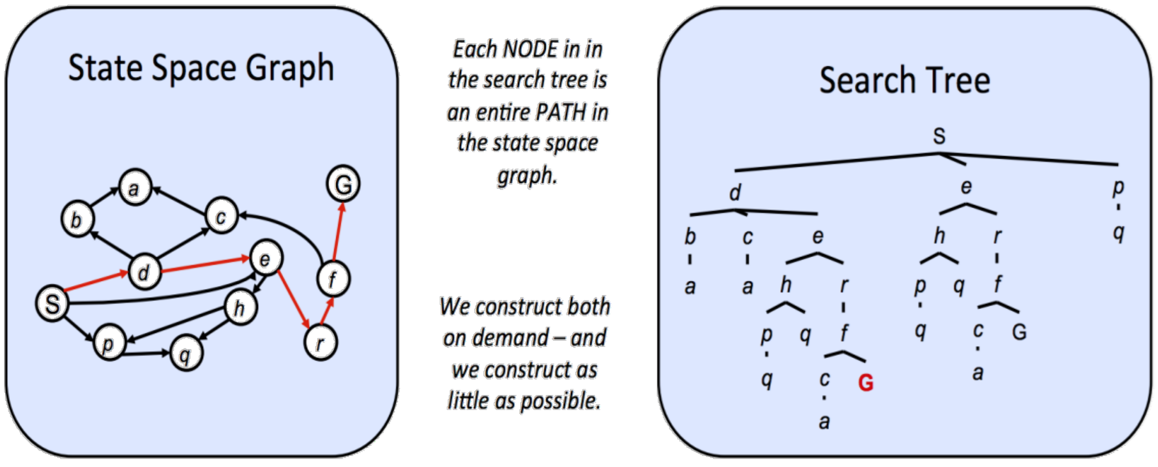


State Space: The set of all states reachable from the initial state. Implicitly defined by the initial state and the successor function, so we have a graph, the state-space graph.
Path: A sequence of states connected by a sequence of actions.
Solution: A path that leads from the initial state to a goal state.
Optimal Solution: Solution with the minimum path cost.

Example
Problem: You are currently in Arad and need to find the fastest way to Bucharest for your flight tomorrow.
Environment: See next slides.
Initial State: Arad
Description of Actions: A function defining the cities that can be reached from the given city.
Goal: Be in Bucharest.
Costs: A function that gives you the costs of traveling from one city to another, based on Speed and/or other criteria.
Solution: Sequence of cities/actions to get to Bucharest from Arad.

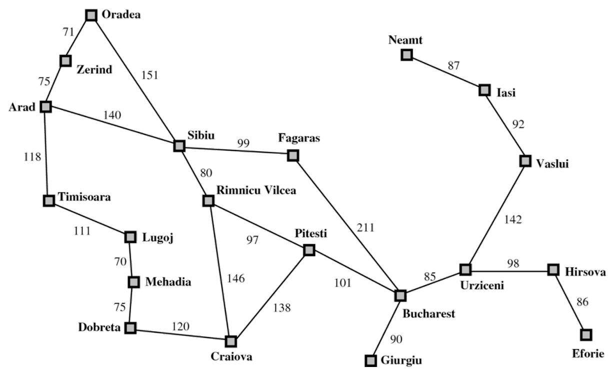
3.2 Tree Search Algorithmus

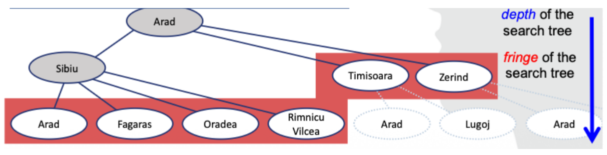
When you treat the state-space graph as a tree you can use simulated iterative exploration of the state space. But this needs a strategy to determine which node is expanded next.


State Space Graph
a
G
b
c
d
e
f
S
h
p
q
r
Each NODE in in the search tree is an entire PATH in the state space graph.
We construct both on demand – and we construct as little as possible.
Search Tree
s
d
e
p
b
c
e
h
r
q
a
a
h
r
p
q
f
p
q
f
q
c
G
q
c
G
a
a

Example: Romania

1. Initial State: start with node Arad
2. Expand node Arad
3. Expand node Sibiu





Fringe

The set of all nodes at the end of all visited paths is called fringe. (other naems are frontier or border)

Depth

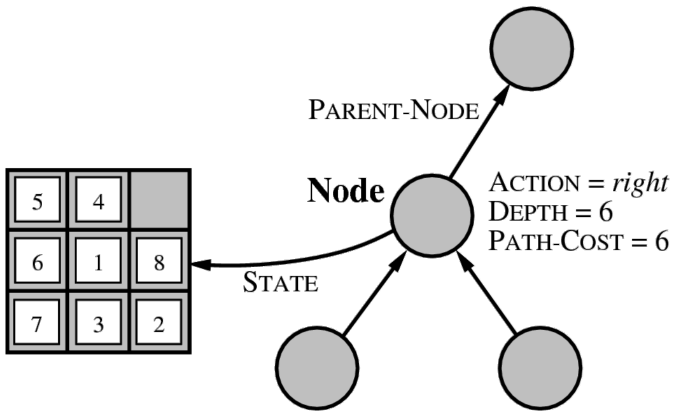
Number of levels in the search tree.

State

Representation of a physical configuration. Describes a specific situation in our environment.

Node

A data structure to present a part of a search tree. it includes a state, a parent node, the taken action, the path costs and the current depth of the tree. Expanding a node, creates new nodes (leaf/successor nodes) and updates the tree.



3.3 Search Strategy

Uninformed Search

Do not have any information except the problem definition.

Breadth-first search (BFS)

Uniform-cost search
Depth-first search (DFS)
Depth-limited search
Iterative deepening

Informed Search

Have additional knowledge about the problem and an idea where to ”look” for solutions.

Greedy Best-first Search
A* Search
Memory-Bounded Heuristic Search

3.3.1 Uninformed Tree Search Strategies

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes*	Yes*	No	Yes, if $l \geq d$	Yes
Time	b^{d+1}	$b^{\lceil C^*/\epsilon \rceil}$	b^m	b^l	b^d
Space	b^{d+1}	$b^{\lceil C^*/\epsilon \rceil}$	bm	bl	bd
Optimal?	Yes*	Yes	No	No	Yes*

Uniform-Cost Search (UCS)

Each node is associated with a fixed cost (nodes can have different costs) and they accumulate over the path within the search. Uniform-Cost Search uses the lowest cumulative cost to find a path, i.e., Next node expanded is the one with the lowest accumulated path cost.

Breadth-First Search BFS

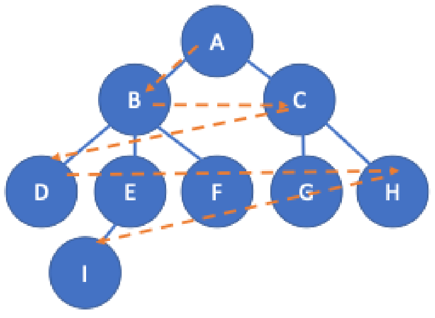
A special case of the UCS, when all costs are equal. It starts at the tree root and explores the tree level by level. Actually, BFS stops as soon as it generates a goal, whereas UCS examines all the nodes at the goal’s depth to see if one has a lower cost.

Completeness: Yes, if each step has a positive cost, otherwise infinite loops are possible. Hence, BFS is also complete.
Complexity: $O(b^d)$ for BFS, $O(b^(1 + floor(OptCost/eps)))$ for UCS, which can be much greater than $O(b^d)$
Optimality: Yes, since nodes expand in increasing order of path costs. In turn BFS is optimal.
BFS and uniform-cost search are often implemented using a priority que ordered by costs.
While conceptually simple, BFS **memory consumption** can be too costly (as it has exponential complexity bound)! In general, exponential-complexity search problems cannot be solved by uninformed methods for any but the smallest instances.

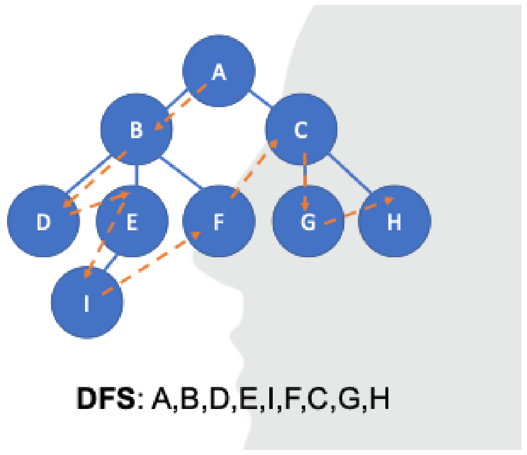
Depth-First Search (DFS)

It starts at the tree root and explores the tree as far as possible along one branch before going step-wise back and explore alternative branches.

Completeness: No, fails in infinite-depth search spaces and spaces with loops. Can be modified to be complete by avoiding repeated states and limit depth. **Time Complexity:** Explores each branch until max depth m (where m is the longest possible path length), i.e., $O(b^m)$. Terrible if $m > d$ (depth of goal node), but may be good in dense settings. **Space Complexity:** Only a branch and their unexpanded siblings has to be stored. Therefore linear complexity, i.e., $O(B * m)$ **Optimality:** No, longer solutions may be found before shorter solutions. Solution could be more expensive then the optimal one.



BFS: A,B,C,D,E,F,G,H,I



DFS: A,B,D,E,I,F,C,G,H

Criteria	**BFS**	**DFS**
Concept	Traversing tree level by level	Traversing tree sub-tree by sub-tree
Data Structure (Queue)	First In First Out (FIFO)	Last In First Out (LIFO)
Time Complexity	O(Vertices + Edges)	O(Vertices + Edges)
Backtracking	No	Yes
Memory	Requires more memory	Less nodes are stored normally (less memory)
Optimality	Yes	Not without modification
Speed	In most cases slower compared to DFS	In most cases faster compared to BFS
When to use	If the target is relatively close to the root node	If the goal state is relatively deep in the tree

Comparison between BFS and DFS algorithms

Depth-limited Search

The depth within the search is limited to l. Nodes with depth $d > l$ are not considered.

Completeness: No
Time Complexity: $O(b^l)$
Space Complexity: $O(b \times l)$
Optimality: No, see DFS

Iterative Deepening Search

Increase l after each failed search, i.e. $l = 1, 2, 3, \ldots$

Completeness: Yes
Time Complexity: first levels has to be search d times $\Rightarrow d \cdot b + (d-1)b^2 + \cdots + 1 \cdot b^d = \sum_{i=1}^{d}(d-i+1) \cdot b^i$
Space Complexity: $O(b \cdot d)$
Optimality: Yes, the shortest path is found.

Bidirectional Search

Perform two search simultaneosly, starting with the root and goal state. Stop if node occurs in both searches.

- Reduction in complexity $b^(d/2) + b^(d/2) << b^d$
- Only possible if actions can be reversed
- Search paths may not meet for depth-first bidirectional search

3.3.2 Informed Tree Search Strategies

The Problem is: Uninformed search algorithms are inefficient.

Greedy Best-first Search

- Using the evaluation function $f(n) = h(n)$ to estimate the cost from node n to goal
- e.g. $h(n) = hSLD(n)$ = straight-line distance from n to Bucharest
- Expand the tree with smallest cost according to $f(n)$, i.e., here the heuristic

Completeness: No, we can get stuck in loops. It is complete in finite state space when we make sure to avoid repeating states.
Time Complexity: Worst case $O(b^m)$, same as DFS but can be improved using good heuristics
Space Complexity: has to keep all nodes in memory, worst case $O(b^m)$
Optimality: No, solution depends on heuristic.

A* Search

An informed tree search algorithm, build on best-first search. Tries to minimize not only the estimated cost $h(n)$ but also the true costs so far $g(n)$.

Best-known form of Best-First Search. The idea behind it is to avoid expanding paths that are already expensive. The goal is to evaluate the complete path cost and not only the remaining costs.
$g(n)$ = cost so far to reach node n
$h(n)$ = estimated cost to get from n to goal
$f(n)$ = estimated cost of path to goal via n
$f(n) = g(n) + h(n)$
Completeness: Yes; Exception: if there are infinitely many nodes with $f(n) \leq f(G)$
Time Complexity: It can be shown that the number of nodes grows exponentially unless the error of the heuristic $h(n)$ is bounded by the logarithm of the value of the actual path cost $h*(n)$, i.e., $|h(n) - h*(n)| \leq O(\log h*(n))$
Space Complexity: Has to keep all nodes in memory.
Optimality: Depends on the heuristic.

3.3.3 Heuristics

Heuristics h

Informally denotes a ”rule of thumb”, i.e. a rule that may be helpful in solving the problem. In tree-search, a heuristic denotes a function h that estimates the remaining costs to reach the goal.

A heuristic is admissible if it never overestimates the cost to reach a goal.

Formally: $h(n) \leq h*(n)$ if $h*(n)$ are the true cost from n to goal

Consistent Heuristics (kind of triangle inequality)

A heuristic is consistent if for every node n and every successor n' generated by any action a it holds that $h(n) \leq c(n, a, an') + h(n')$. Thus, a heuristic is consistent if, when going from neighboring nodes a to b, the heuristic difference/step cost never overestimates the actual step cost.

If there were a route from n to the goal that was cheaper than $h(n)$, that would violate the property that $h(n)$ is a lower bound on the cost to reach the goal (kind of triangle inequality).
Lemma 1: Every consistent heuristic is 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.
Looking for relaxed problems is a good strategy for inventing admissible heuristics.

Dominance

If h1 and h2 are both admissible heuristics and $h2(n) \leq h1(n)$ for all n, then $h2$ dominates $h1$.

If $h2$ dominates $h1$ it will perform better as it will always be closer to the optimal heuristic $h*$.
Why is this important? Search algorithms expand every node with $f(n) < C*$ vs. $h(n) < C* - g(n)$ Thus, dominant heuristics result in less expansion.

Combining Heuristics Suppose we have a collection of admissible heuristics $h1(n), h2(n), \ldots, hm(n)$, but none of them dominates the others.
Theorem (Combining admissible heuristics): If $h1$ and $h2$ are two admissible heuristics than $h(n) = max h1(n), h2(n), \ldots, hm(n)$ is also admissible and dominates $h1$ and $h2$.

Optimality of A* for Admissible Heuristics
Let C^* be the cost of an optimal solution (an optimal goal G). Assume h is admissible, i.e. $h(n) \leq h^*(n)$ for every node n, where $h^*(n)$ is the true cost from n to a goal.

Proof (tree-search style). Assume for contradiction that A* returns a suboptimal goal G_2 with cost $g(G_2) > C^*$. Consider any node n on an optimal path from the start to the optimal goal G that is currently in the fringe and has not yet been expanded. For that node

$$\begin{aligned} f(n) &= g(n) + h(n) \\ &\leq g(n) + h^*(n) \quad \text{(since } h \text{ is admissible)} \\ &= C^*. \end{aligned}$$

But

$$f(G_2) = g(G_2) > C^*.$$

Hence $f(n) \leq C^* < f(G_2)$, so A* would expand n (or another node with $f \leq C^*$) before selecting G_2. Therefore A* cannot return a suboptimal goal before expanding all nodes with $f \leq C^*$, contradicting the assumption that it returns G_2. Thus A* with an admissible heuristic is optimal.

Proof (informal, written form). Assume for contradiction that A* selects a path p to a goal but some other path p' is actually the shortest path. Just before A* chooses p from the fringe, some prefix p'' of p' must be on the fringe. Since A* chose p before p'', we have $f(p) \leq f(p'')$. Because p is a goal and h is admissible, $h(p) = 0$, so $f(p) = \text{cost}(p)$. Also, admissibility implies $\text{cost}(p'') + h(p'') \leq \text{cost}(p')$. Combining these gives $\text{cost}(p) \leq \text{cost}(p')$, contradicting that p' is the shorter path. Hence A* is optimal.

Three alternatives to A*: Memory-Bounded Heuristic Search **Problem:** A* (and Best-First search) has a space problem

1. Iterative Deepening A* (IDA*)
    - like iterative deepening
    - cutoff information is the f-cost $(g + h)$ instead of depth
2. Recursive best-first search (RBFS)
    - recursive algorithm that attempts to mimic standard best-first search with linear space.
    - keeps track of the f-value to the best alternative

  - patch available from any ancestor of current node and heuristic evaluations are upated with results of successors

3. (Simple Memory-Bounded A* ((S)MA)*)
   - drop the worst leaf node when memory is full
   - its value will be updated to its parent
   - may need to be researched later

The failure to detect repeated states can turn a linear problem into an exponential one.