What is AI?

Definition of Artificial Intelligence

- AI is concerned with designed of intelligence in artificial devices.

- The definitions of AI can be categorised in two dimensions:
  - Thought processes or reasoning
  - Behaviour

- There can be two paradigms in AI
  - Human-like performance
  - Ideal performance – rational way. A system is rational if it does right thing, given what it knows.

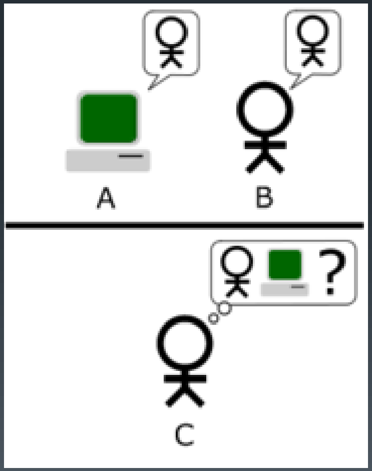
Systems that think like humans	Systems that think rationally
Systems that act like humans	Systems that act rationally

Different Types of Artificial Intelligence

- Modeling exactly how humans actually think
  - cognitive models of human reasoning
- Modeling exactly how humans actually act
  - models of human behavior (what they do, not how they think)
- Modeling how ideal agents “should think”
  - models of “rational” thought (formal logic) note: humans are often not rational!
- Modeling how ideal agents “should act”
  - rational actions but not necessarily formal rational reasoning i.e., more of a black-box/engineering approach

Acting Humanly: The Turing Test Approach

- Proposed by Alan Turing
- (Human) judge communicates with a human and a machine over text-only channel.
- Both human and machine try to act like a human.
- Judge tries to tell which is which



- For the computer to successfully fool the human judge it has possess following capabilities:
  - Natural language processing: Ability to understand and successfully communicate in a language
  - Knowledge representation: To store what is knows or hears
  - Automated reasoning: To use the stored information to answer questions and to draw new conclusions
  - Machine Learning: Adapt to new circumstances an to detect and extrapolate patterns.

Thinking humanly: The cognitive modelling approach

- Based on human-way of thinking.
- Need to get into actual working of human minds
- Two ways
  - Through introspection
  - Through psychological experiments
- Cognitive science: Combination of AI models and psychology

Thinking Rationally: The "Laws of Thought" Approach

- The emphasis is on correct inferences
- Involves reasoning logically to the conclusion that a given action will achieve one's goals and than to act on that conclusion.
- Two main obstacles:
  - Difficult to state all informal knowledge in formal terms
  - Big difference between solving a problem in principle and doing so in practice. A set of few facts can also exhaust all computational resources.

Acting rationally: The rational agent approach

- An agent is just something that acts
- An agent have additional attributes such as operating under autonomous control, perceiving their environment, adapting to changes and being capable to take some other’s goals.
- A rational agent is one that acts so as to the achieve best outcome, or when there is uncertainty, the best expected outcome.
- Two advantages of Rational agent design over Laws of Thought approach
  - More general
  - More amenable to scientific development.

A brief history

- What happened after WWII?
  - 1943: Warren Mc Culloch and Walter Pitts: a model of artificial boolean neurons to perform computations.
  - First steps toward connectionist computation and learning (Hebbian learning).
  - Marvin Minsky and Dann Edmonds (1951) constructed the first neural network computer
  - 1950: Alan Turing’s “Computing Machinery and Intelligence” book
    - First complete vision of AI.

- The birth of AI (1956)
  - Darmouth Workshop bringing together top minds on automata theory, neural nets and the study of intelligence.
  - Allen Newell and Herbert Simon: The logic theorist (first nonnumerical thinking program used for theorem proving). For the next 20 years the field was dominated by these participants.
- Great expectations (1952-1969)
  - Newell and Simon introduced the General Problem Solver.
    - Imitation of human problem-solving
  - Arthur Samuel (1952-)investigated game playing (checkers) with great success.
  - John McCarthy(1958-) :
    - Inventor of Lisp (second-oldest high-level language)
    - Logic oriented, Advice Taker (separation between knowledge and reasoning)

- AI revival through knowledge-based systems (1969-1970).

- General-purpose vs. domain specific
  - E.g. the DENDRAL project
    - First successful knowledge intensive system.
  - Expert systems
  - MYCIN to diagnose blood infections
    - Introduction of uncertainty in reasoning.
- Increase in knowledge representation research.
  - Logic, frames, semantic nets, …

- AI becomes an industry (1980 - present)
- Connectionist revival (1986 - present)
  - Parallel distributed processing (RumelHart and McClelland, 1986); backpropagation.

- AI becomes a science (1987 - present)
    - In speech recognition: hidden markov models
    - In neural networks
    - In uncertain reasoning and expert systems: Bayesian network formalism

- The emergence of intelligent agents (1995 - present)
  - The whole agent problem:
    - "How does an agent act/behave embedded in real environments with continuous sensory inputs"

State of the art

- Deep Blue defeated the reigning world chess champion Garry Kasparov in 1997
- ALVINN: No hands across America (driving autonomously 98% of the time from Pittsburgh to San Diego)
- DART:During the 1991 Gulf War, US forces deployed an AI logistics planning and scheduling program that involved up to 50,000 vehicles, cargo, and people
- NASA's on-board autonomous planning program controlled the scheduling of operations for a spacecraft
- Proverb solves crossword puzzles better than most humans

Intelligent Systems in Your Everyday Life

- Post Office
  - automatic address recognition and sorting of mail
- Banks
  - automatic check readers, signature verification systems
  - automated loan application classification
- Telephone Companies
  - automatic voice recognition for directory inquiries
- Credit Card Companies
  - automated fraud detection
- Computer Companies
  - automated diagnosis for help-desk applications
- Netflix:
  - movie recommendation
- Google:
  - Search Technology

Intelligent Agents

Agent

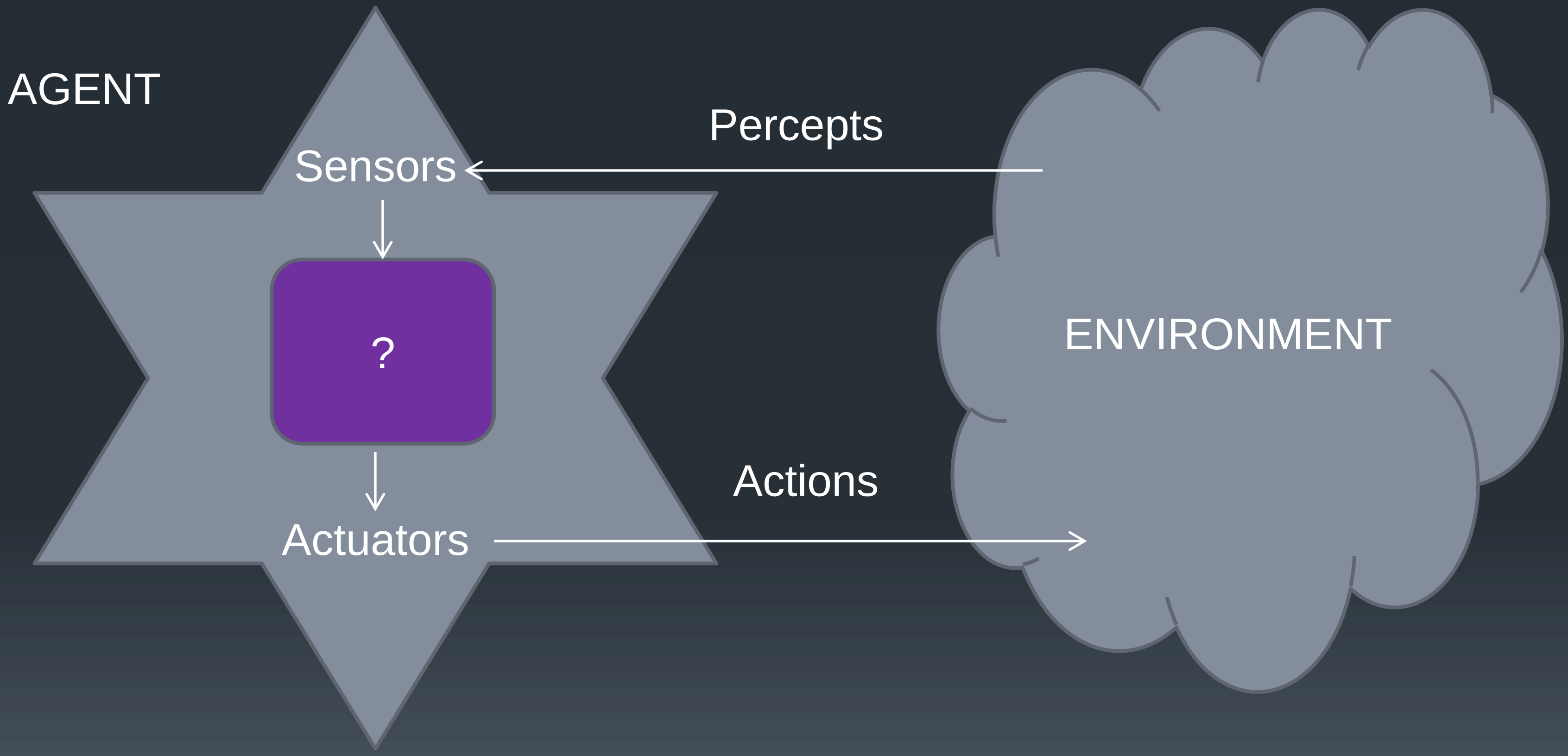
- An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators.
- An agent operates in an environment.
- The agent receives percepts from the environment and the agent acts and its actions can change the environment.
- The agent uses its various sensory organs so depending upon the sensors that the agent has for example the agent may be able to see if the agent has a camera, the agent may be able to hear if it has a sonar sensor and so agent can see or hear or accept different inputs from the environment.

Percept

- Percept refers to the agent’s perceptual inputs at any given instant
- The complete set of inputs at a given time the agent gets is called its percept.
- The input can be from the keyboard or through its various sensors.
- The sequence of percepts may be the current percept or may be all the percepts that the agent has perceived so far can influence the actions of an agent.
- An agent’s percept sequence is the complete history of everything the agent has ever perceived.

Actuators

- The agent can change the environment through effectors or actuators.
- An operation which involves an actuator is called an action.
- So, agent can take action in an environment through the output device or through the different actuators that it might be having. These actions can be grouped into action sequences.
- Actuators are sometimes referred to as effectors.


AGENT
Sensors
?
Actuators
Percepts
Actions
ENVIRONMENT

Agent Function

- Agent Function describes the agent’s behaviour that maps any given percept sequence to an action.
- Internally, the agent function of an artificial agent will be implemented by an agent program.
- Agent function is abstract mathematical description, while agent program is a concrete implementation, running on the agent architecture.

Good behaviour: Concept of Rationality

- A rational agent is one that does the right thing.
- An approximation – the right action is the one that will cause the agent to be most successful.
- Question - how and when to evaluate the agent’s success?

Performance Measure

- The term performance measure for the how—the criteria that determine how successful an agent is.
- Objective performance measure imposed by some authority - a standard of what it means to be successful in an environment and use it to measure the performance of agents.

- As an example, consider the case of an agent that is supposed to vacuum a dirty floor.
- A plausible performance measure would be the amount of dirt cleaned up in a single eight-hour shift.
- A more sophisticated performance measure would factor in the amount of electricity consumed and the amount of noise generated as well.
- A third performance measure might give highest marks to an agent that not only cleans the floor quietly and efficiently, but also finds time to go windsurfing at the weekend

- The when of evaluating performance is also important.
- If we measured how much dirt the agent had cleaned up in the first hour of the day, we would be rewarding those agents that start fast (even if they do little or no work later on), and punishing those that work consistently.
- Thus, we want to measure performance over the long run, be it an eight-hour shift or a lifetime

Rationality VS Omniscience

- An omniscient agent knows the actual outcome of its actions, and can act accordingly; but omniscience is impossible in reality.
- We cannot blame an agent for failing to take into account something it could not perceive, or for failing to take an action (such as repelling the cargo door) that it is incapable of taking.

- In summary, what is rational at any given time depends on four things:
  - The performance measure that defines degree of success.
  - Everything that the agent has perceived so far - the percept sequence.
  - What the agent knows about the environment.
  - The actions that the agent can perform.

Rational agent

- Rational agent: For each possible percept sequence, an rational agents should do whatever action is expected to maximize its performance measure, on the basis of the evidence provided by the percept sequence and whatever built-in knowledge the agent has.

Mapping

- Mapping is a list of relations between percept sequences to its actions.
- A mapping can correctly describe an agent by trying out all possible percept sequences and recording which actions the agent does in response.
- And if mappings describe agents, then ideal mappings describe ideal agents
- Specifying which action an agent ought to take in response to any given percepts sequence provides a design for an ideal agent.

Autonomy

- If the agent’s actions are based completely on built-in knowledge, such that it need pay no attention to its percepts, then we say that the agent lacks autonomy.
- An agent’s behavior can be based on both its own experience and the built-in knowledge used in constructing the agent for the particular environment in which it operates.
- A system is autonomous to the extent that its behavior is determined by its own experience.

STRUCTURE OF INTELLIGENT AGENTS

- The job of AI is to design the agent program: a function that implements the agent mapping from percepts to actions.
- We assume this program will run on some sort of computing device, which we will call the architecture to be one that the architecture will accept and run.
- Architecture might also include software that provides a degree of insulation between the raw computer and the agent program, so that we can program at a higher level.
- The relationship among agents, architectures, and programs can be summed up as follows:

  agent = architecture + program

Agent programs

- Agent programs, initially, will have a very simple form and will use some internal data structures that will be updated as new percepts arrive.
- These data structures are operated on by the agent’s decision-making procedures to generate an action choice, which is then passed to the architecture to be executed.

- The agent program receives only a single percept as its input. It is up to the agent to build up the percept sequence in memory, if it so desires
- The goal or performance measure is not part of the skeleton program. This is because the performance measure is applied externally to judge the behavior of the agent, and it is often possible to achieve high performance without explicit knowledge of the performance measure

- function SKELETON-AGENT(percept) returns action
  - static: memory, the agent’s memory of the world

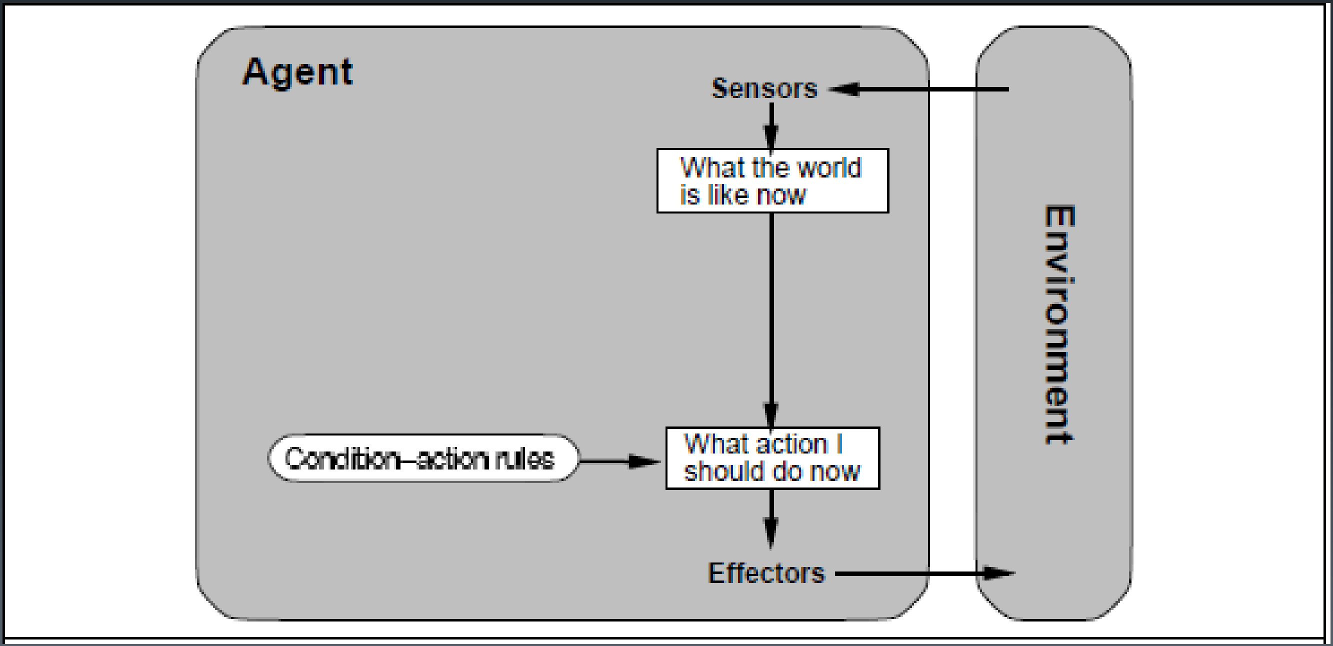
  - memory UPDATE-MEMORY(memory, percept)
  - action CHOOSE-BEST-ACTION(memory)
  - memory UPDATE-MEMORY(memory, action)
- return action

Agent Types

- Simple reflex agents
- Agents that keep track of the world
- Goal-based agents
- Utility-based agents
- Learning Based Agent

Simple reflex agents

- Follow condition–action rule
- Eg: “The car in front is braking”; then this triggers some established connection in the agent program to the action “initiate braking”



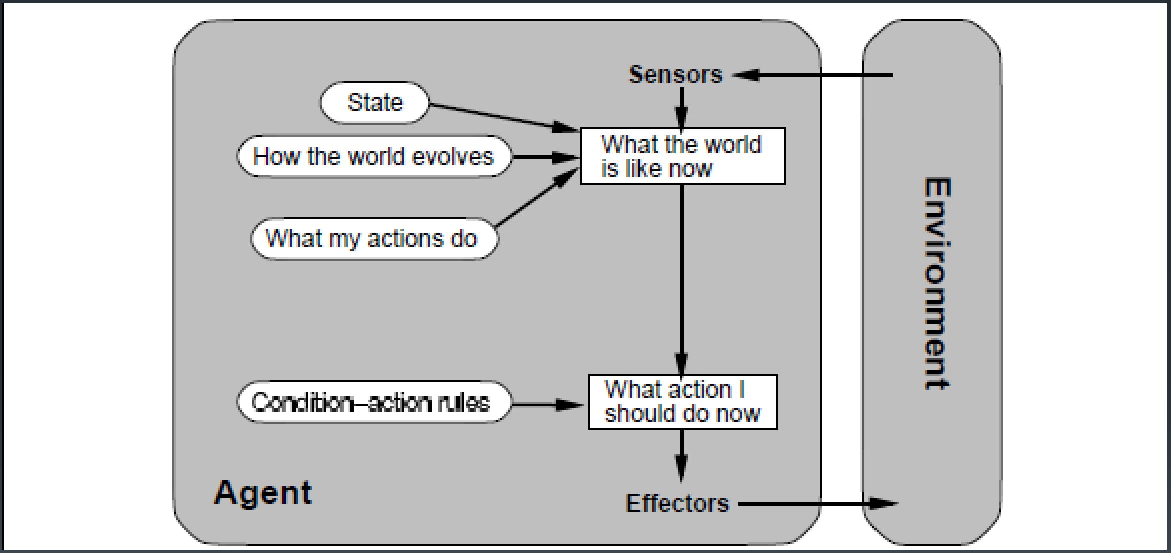
- function SIMPLE-REFLEX-AGENT(percept) returns action
  - static: rules, a set of condition-action rules
  - State<-INTERPRET-INPUT(percept)
  - rule <- RULE-MATCH(state, rules)
  - action <- RULE-ACTION[rule]
- return action

- The INTERPRET-INPUT function generates an abstracted description of the current state from the percept, and the RULE-MATCH function returns the first rule in the set of rules that matches the given state description.
- Although such agents can be implemented very efficiently, their range of applicability is very narrow

Agents that keep track of the world

- Problem may arise because the sensors do not provide access to the complete state of the world.
- In such cases, the agent may need to maintain some internal state information in order to distinguish between world states that generate the same perceptual input but nonetheless are significantly different.
- Here, “significantly different” means that different actions are appropriate in the two states.

- Updating this internal state information as time goes by requires two kinds of knowledge to be encoded in the agent program.
- First, we need some information about how the world evolves independently of the agent—for example, that an overtaking car generally will be closer behind than it was a moment ago.
- Second, we need some information about how the agent's own actions affect the world

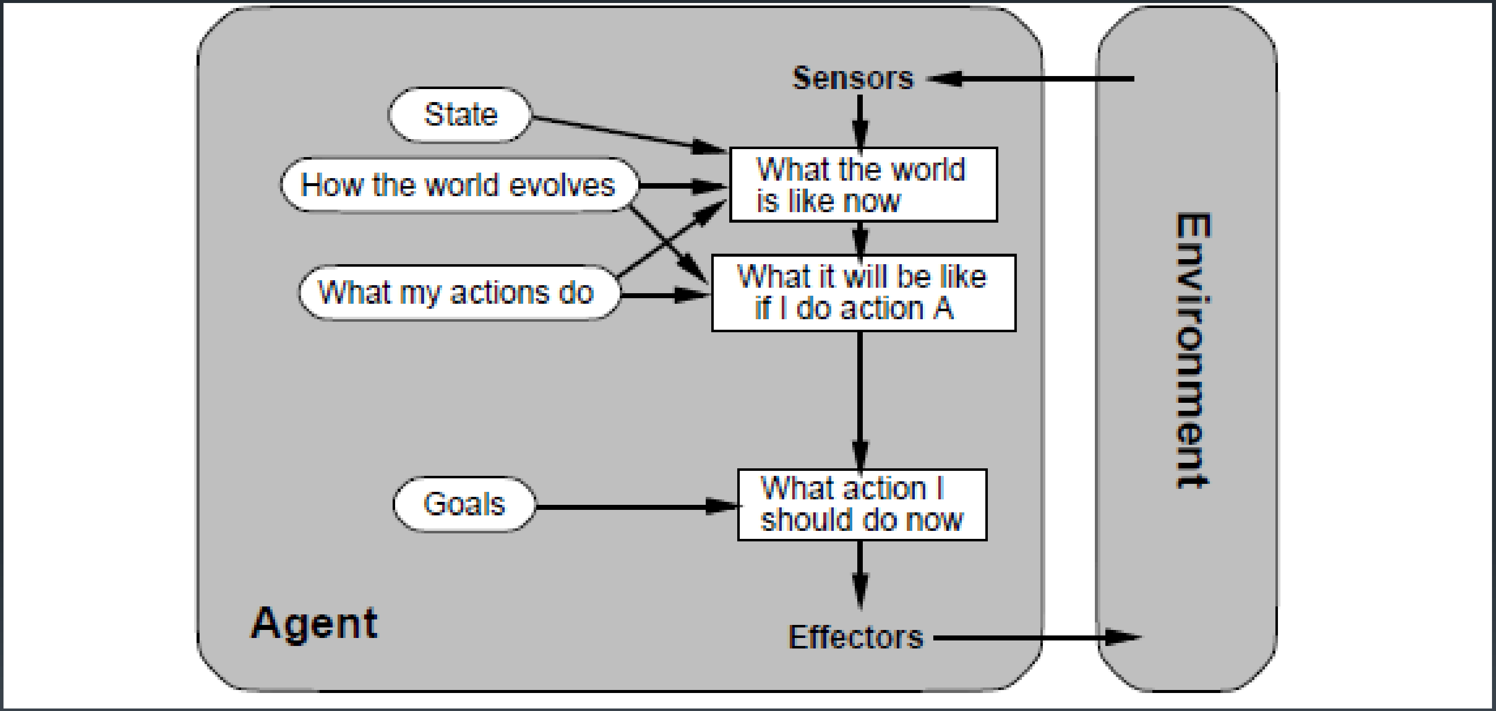

Sensors
State
How the world evolves
What my actions do
What the world is like now
Condition–action rules
What action I should do now
Agent
Effectors
Environment

- function REFLEX-AGENT-WITH-STATE(percept) returns action
  - static: state, a description of the current world state
  - rules, a set of condition-action rules
  - state <- UPDATE-STATE(state, percept)
  - rule <- RULE-MATCH(state, rules)
  - action <- RULE-ACTION[rule]
  - state <- UPDATE-STATE(state, action)
- return action

Goal-based agents

- The agent program can combine this with information about the results of possible actions (the same information as was used to update internal state in the reflex agent) in order to choose actions that achieve the goal.
- Sometimes this will be simple, when goal satisfaction results immediately from a single action; sometimes, it will be more tricky, when the agent has to consider long sequences of twists and turns to find a way to achieve the goal.

- Although the goal-based agent appears less efficient, it is far more flexible.
- Of course, the goal-based agent is also more flexible with respect to reaching different destinations. Simply by specifying a new destination, we can get the goal-based agent to come up with a new behavior.

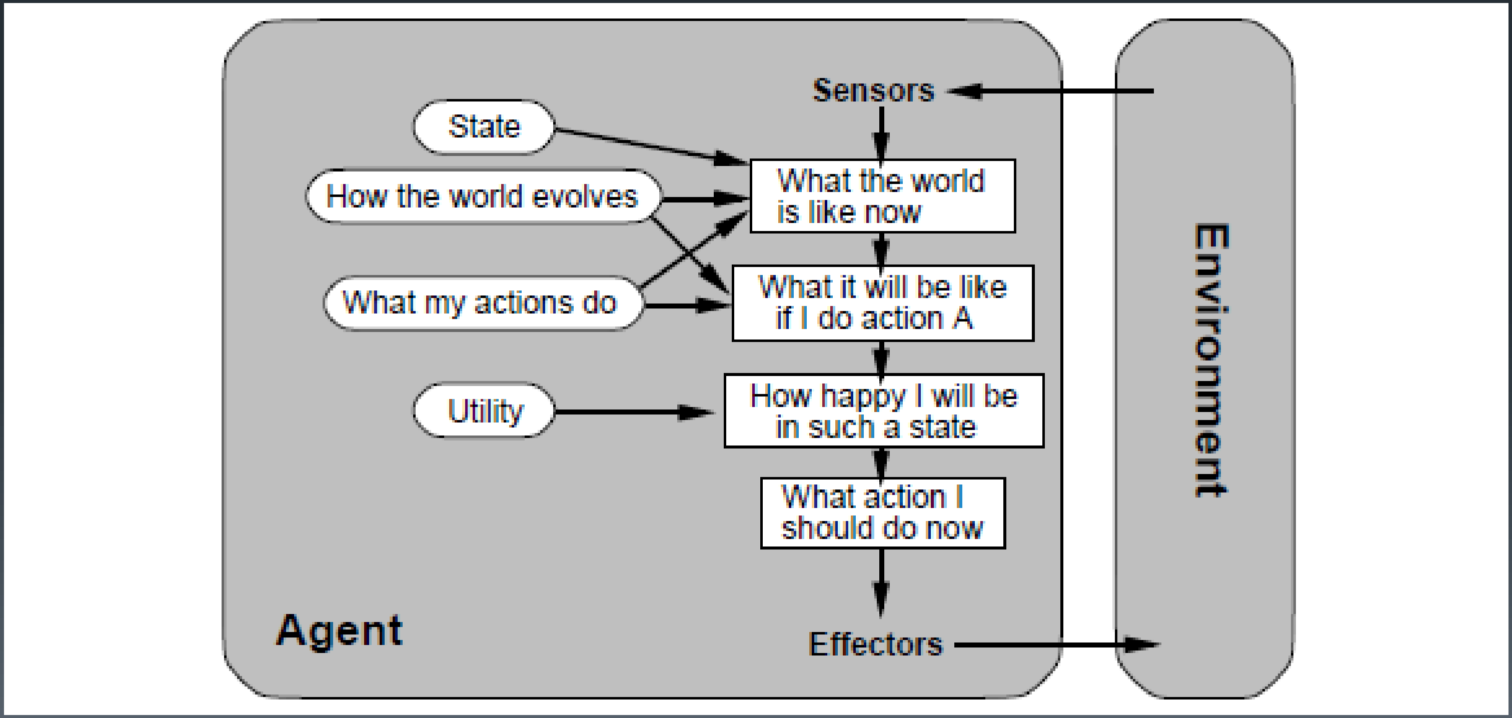

Sensors
State
How the world evolves
What my actions do
What the world is like now
What it will be like if I do action A
Goals
What action I should do now
Agent
Effectors
Environment

Utility-based agents

- Goals just provide a crude distinction between “happy” and “unhappy” states, whereas a more general performance measure should allow a comparison of different world states (or sequences of states) according to exactly how happy they would make the agent if they could be achieved.
- Because “happy” does not sound very scientific, the customary terminology is to say that if one world state is preferred to another, then it has higher utility for the agent

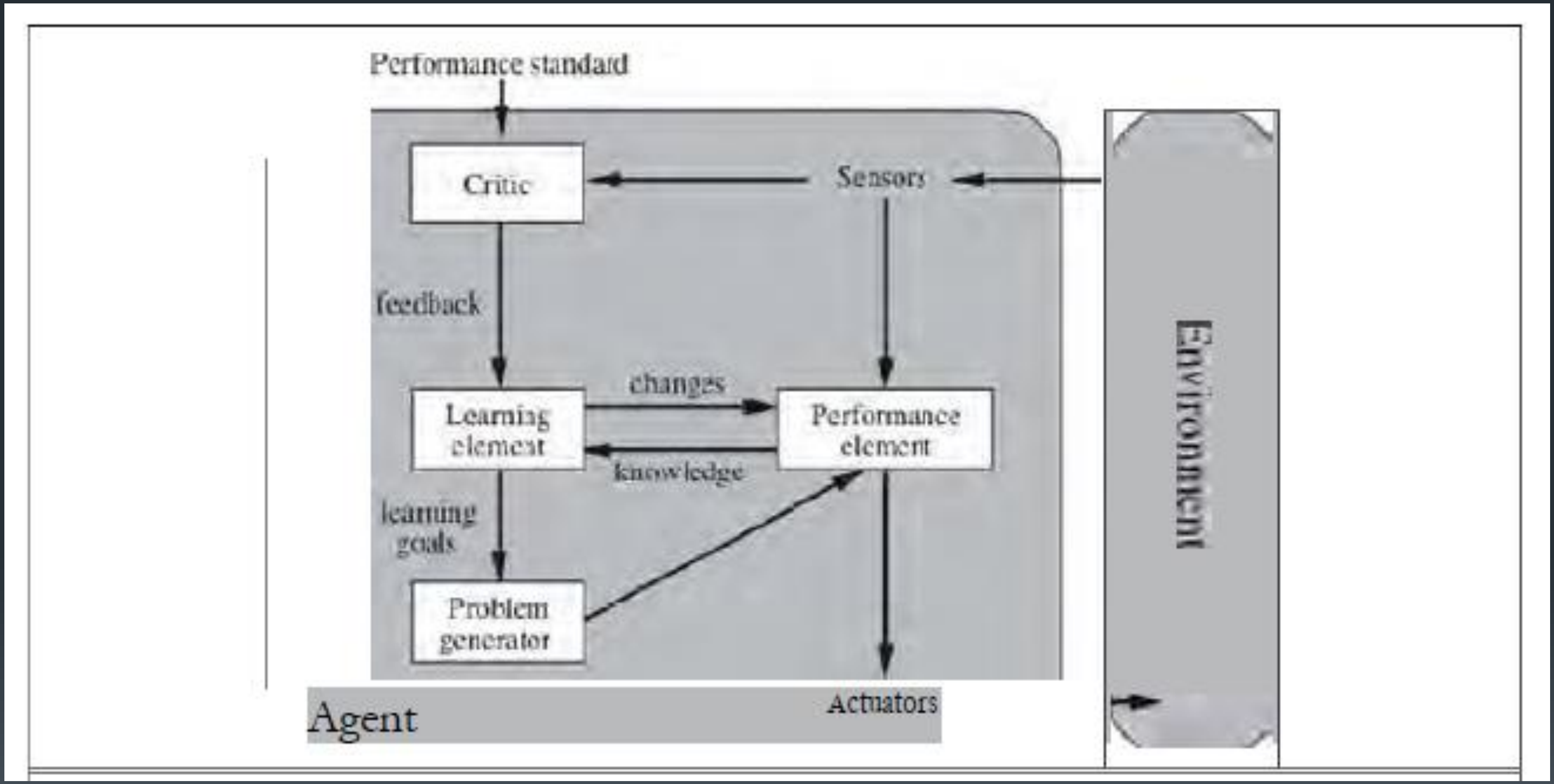
- Utility is therefore a function that maps a states onto a real number, which describes the associated degree of happiness.
- A complete specification of the utility function allows rational decisions in two kinds of cases where goals have trouble.
  - First, when there are conflicting goals, only some of which can be achieved (for example, speed and safety), the utility function specifies the appropriate trade-off.
  - Second, when there are several goals that the agent can aim for, none of which can be achieved with certainty, utility provides a way in which the likelihood of success can be weighed up against the importance of the goals.

- An agent that possesses an explicit utility function therefore can make rational decisions, but may have to compare the utilities achieved by different courses of actions.


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

Learning Based Agent

- Divided into four conceptual components:
  - Learning element: Responsible for making improvement
  - Performance element: Responsible for selecting external action
  - Critic: tells the learning element how well the agent is doing with respect to a fixed performance standar
  - Problem generator. It is responsible for suggesting actions that will lead to new and informative experiences


Performance standard
Critic
Sensors
feedback
changes
Learning element
Performance element
knowledge
learning goals
Problem generator
Agent
Actuators
Environment

- The performance element is what we have previously considered to be the entire agent: it takes in percepts and decides on actions.
- The learning element uses feedback from the critic on how the agent is doing and determines how the performance element should be modified to do better in the future.

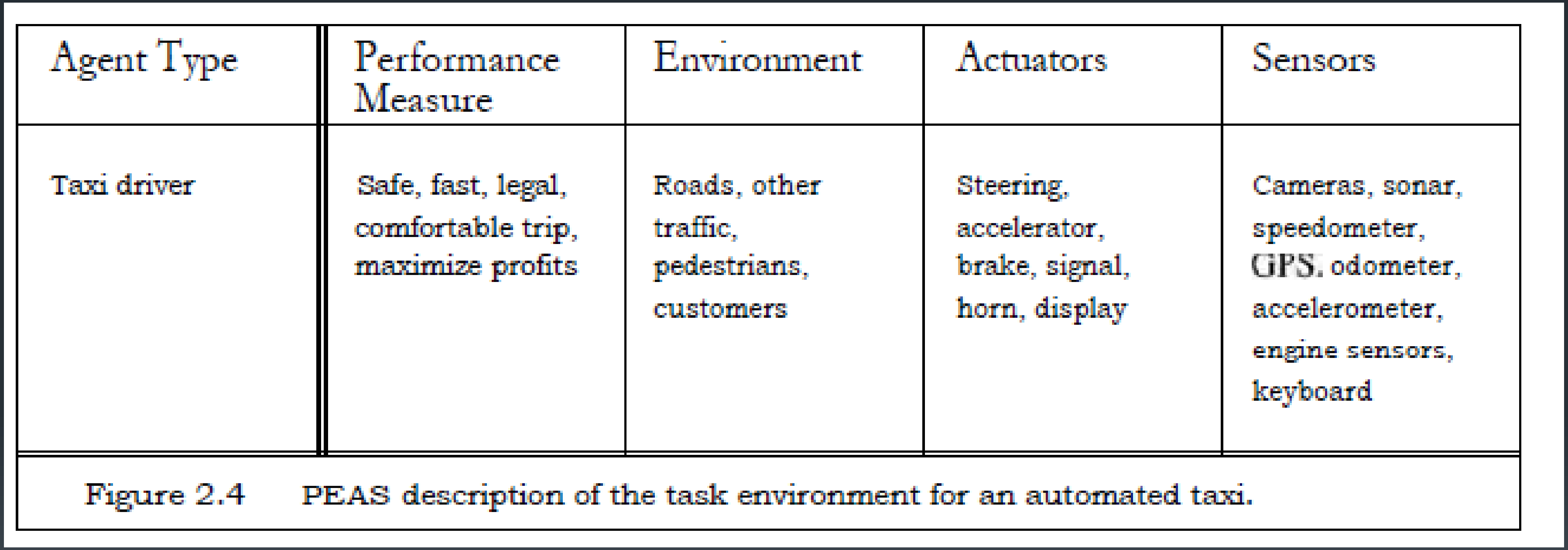
THE NATURE OF ENVIRONMENTS

Task environments

- Task environments, which are essentially the "problems" to which rational agents are the "solutions."
- In simple vacuum-cleaner agent, we had to specify the performance measure, the environment, and the agent's actuators and sensors. It is grouped all under the heading of the task environment.

PEAS (Performance, Environment, Actuators, Sensors)

- Eg. Automated Taxi



Agent Type	Performance Measure	Environment	Actuators	Sensors
Taxi driver	Safe, fast, legal, comfortable trip, maximize profits	Roads, other traffic, pedestrians, customers	Steering, accelerator, brake, signal, horn, display	Cameras, sonar, speedometer, GPS, odometer, accelerometer, engine sensors, keyboard

Figure 2.4 PEAS description of the task environment for an automated taxi.

Properties of task environments

- Fully observable vs. partially observable:
  - If an agent's sensors give it access to the complete state of the environment at each point in time, then we say that the task environment is fully observable.
  - A task environment is effectively fully observable if the sensors detect all aspects that are relevant to the choice of action; relevance, in turn, depends on the performance measure.

- Fully observable environments are convenient because the agent need not maintain any internal state to keep track of the world.
- An environment might be partially observable because of noisy and inaccurate sensors or because parts of the state are simply missing from the sensor data—for example, a vacuum agent with only a local dirt sensor cannot tell whether there is dirt in other squares

- Single agent vs. multiagent:
  - An agent solving a crossword puzzle by itself is clearly in a single-agent environment, whereas an agent playing chess is in a two-agent environment.
  - Described which entities must be viewed as agents. Does an agent A (the taxi driver for example) have to treat an object B (another vehicle) as an agent or can it be treated merely as an object behaving according to the laws of physics?
  - The key distinction is whether B's behavior is best described as maximizing a performance measure whose value depends on agent A's behavior. For example, in chess, the opponent entity B is trying to maximize its performance measure, which, by the rules of chess, minimizes agent as performance measure.

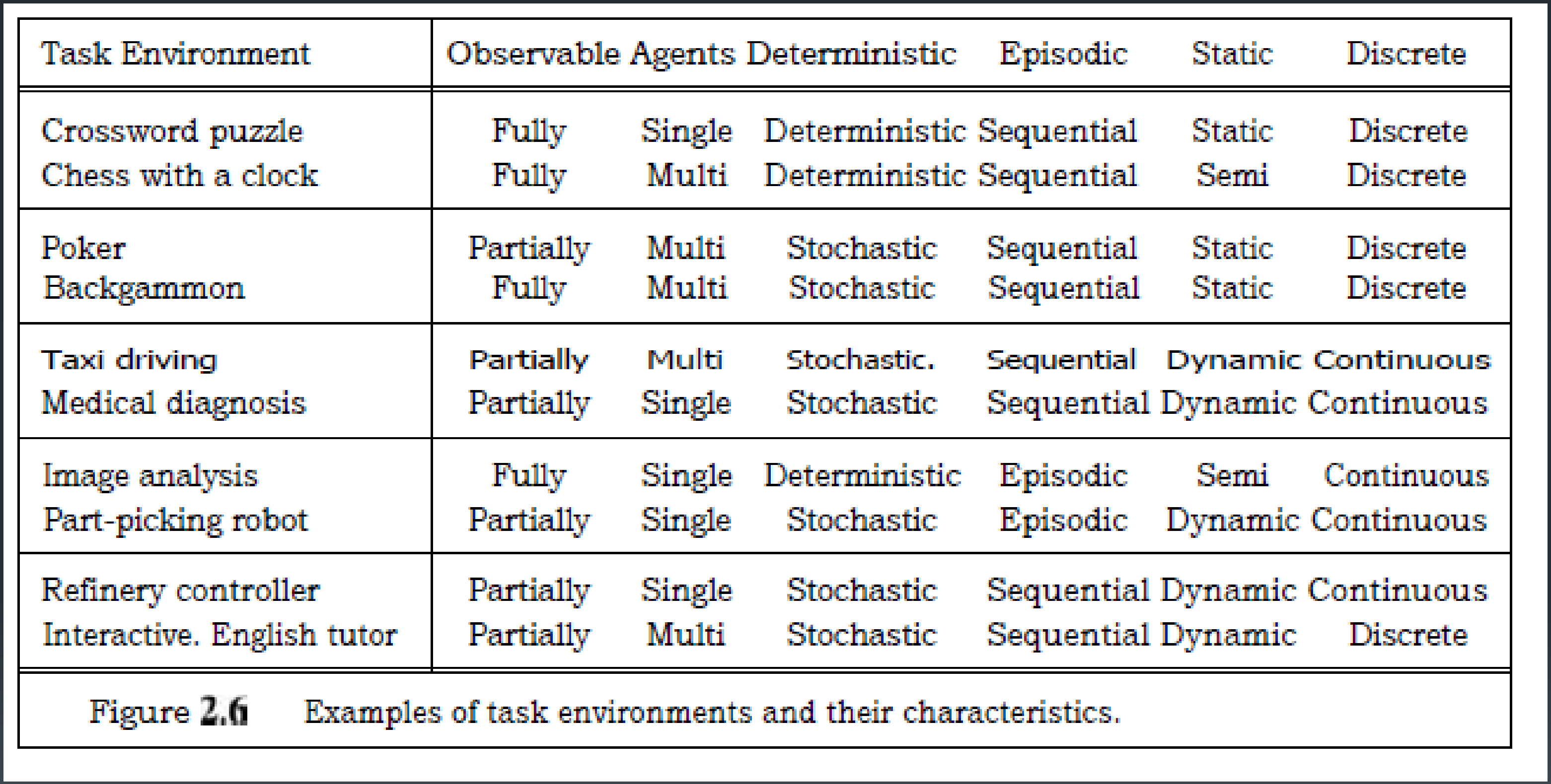
- Deterministic vs. stochastic.
  - If the next state of the environment is completely determined by the current state and the action executed by the agent, then we say the environment is deterministic.
  - If the environment is partially observable, however, then it could appear to be stochastic.

- Episodic vs. sequential:
  - In an episodic task environment, the agent's experience is divided into atomic episodes.
  - In each episode the agent receives a percept and then performs a single action.
  - The next episode does not depend on the actions taken in previous episodes.
  - For example, an agent that has to spot defective parts on an assembly line bases each decision on the current part, regardless of previous decisions
  - In sequential environments, on the other hand, the current decision could affect all future decisions. Chess and taxi driving arc sequential: in both cases, short-term actions can have long-term consequences

- Static vs. dynamic: If the environment can change while an agent is deliberating, then we say the environment is dynamic for that agent; otherwise, it is static.
- Static environments are easy to deal with because the agent need not keep looking at the world while it is deciding on an action, nor need it worry about the passage of time.
- Dynamic environments, on the other hand. are continuously asking the agent what it wants to do; if it hasn't decided yet. that counts as deciding to do nothing.
- If the environment itself does not change with the passage of time but the agent's performance score does, then we say the environment is semidynamic.
- Taxi driving is clearly dynamic: the other cars and the taxi itself keep moving while the driving algorithm dithers about what to do next.
- Chess, when played with a clock, is semidynamic. Crossword puzzles are static.

- Discrete vs. continuous:
  - The discrete/continuous distinction applies to the state of the environment, to the way time is handled, and to the percepts and actions of the agent.
  - For example, the chess environment has a finite number of distinct states (excluding the clock), Chess also has a discrete set of percepts and actions.
  - Taxi driving is a continuous-state and continuous-time problem: the speed and location of the taxi and of the other vehicles sweep through a range of continuous values and do so smoothly over time.
  - Taxi-driving actions are also continuous (steering angles, etc.). Input from digital cameras is discrete, strictly speaking, but is typically treated as representing continuously varying intensities and locations.

- Known vs. unknown:
  - This distinction refers not to the environment itself but to the agent's or designer's state of knowledge about the "laws of physics" of the environment.
  - In a known environment, the outcomes (or outcome probabilities if the environment is stochastic) for all actions are given.
  - If the environment is unknown, the agent will have to learn how it works in order to make good decisions.
  - Note that the distinction between known and unknown environments is not the same as the one between fully and partially observable environments. It is quite possible for a known environment to be partially observable—for example, in solitaire card games, I know the rules but am still unable to see the cards that have not yet been turned over



Task Environment	Observable	Agents	Deterministic	Episodic	Static	Discrete
Crossword puzzle	Fully	Single	Deterministic	Sequential	Static	Discrete
Chess with a clock	Fully	Multi	Deterministic	Sequential	Semi	Discrete
Poker	Partially	Multi	Stochastic	Sequential	Static	Discrete
Backgammon	Fully	Multi	Stochastic	Sequential	Static	Discrete
Taxi driving	Partially	Multi	Stochastic.	Sequential	Dynamic	Continuous
Medical diagnosis	Partially	Single	Stochastic	Sequential	Dynamic	Continuous
Image analysis	Fully	Single	Deterministic	Episodic	Semi	Continuous
Part-picking robot	Partially	Single	Stochastic	Episodic	Dynamic	Continuous
Refinery controller	Partially	Single	Stochastic	Sequential	Dynamic	Continuous
Interactive. English tutor	Partially	Multi	Stochastic	Sequential	Dynamic	Discrete

Figure **2.6** Examples of task environments and their characteristics.

Problem Solving by searching

PROBLEM-SOLVING AGENTS

- Imagine an agent in the city of Arad, Romania, enjoying a touring holiday.
- The agent's performance measure contains many factors: it wants to improve its suntan, improve its Ro- manian, take in the sights, enjoy the nightlife (such as it is), avoid hangovers, and so on.
- The decision problem is a complex one involving many tradeoffs and careful reading of guide-books.
- Now, suppose the agent has a nonrefundable ticket to fly out of Bucharest the following day.
- In that case, it makes sense for the agent to adopt the goal of getting to Bucharest.

- Goal formulation, based on the current situation and the agent's performance measure, is the first step in problem solving.
- Goals help organize behavior by limiting the objectives that the agent is trying to achieve and hence the actions it needs to consider.
- A goal can be a set of world states—exactly those states in which the goal is satisfied.
- `The agent's task is to find out how to act, now and in the future, so that it reaches a goal state.

- Problem formulation is the process of deciding what actions and states to consider, given a goal.
- If it were to consider actions at the level of "move the left foot forward an inch" or "turn the steering wheel one degree left." the agent would probably never find its way out of the parking lot, let alone to Bucharest, because at that level of detail there is too much uncertainty in the world and there would be too many steps in a solution.
- The agent should consider actions at the level of driving from one major town to another. Each state therefore corresponds to being in a particular town.

- Sequence of actions
  - An agent with several immediate options of unknown value can decide what to do by first examining future actions that eventually lead to states of known value.
  - Example: Travelling by road to Bucharest
    - Case 1: Without a map
    - Case 2: With a map
  - To be more specific about what we mean by "examining future actions," we have to be more specific about properties of the environment.

- We assume that the environment is observable, so the agent always knows the current state.
- We also assume the environment is discrete. so at any given state there are only finitely many actions to choose from.
- We will assume the environment is known, so the agent knows which states are reached by each action. (Having an accurate map suffices to meet this condition for navigation problems.)
- Finally, we assume that the environment is deterministic, so each action has exactly one outcome.
- Under these assumptions, the solution in any problem is a, fired sequence of actions.

- Search
  - The process of looking for a sequence of actions that reaches the goal is called search.
  - A search algorithm takes a problem as input and returns a solution in the form of an action sequence.
  - Once a solution is found, the actions it recommends can be carried out. This is called the execution phase .
  - Thus, we have a simple "formulate, search, execute" design for the agent,

- After formulating a goal and a problem to solve. the agent calls a search procedure to solve it.
- It then uses the solution to guide its actions, doing whatever the solution recommends as the next thing to do—typically, the first action of the sequence—and then removing that step from the sequence.
- Once the solution has been executed, the agent will formulate a new goal

- Open-loop system
  - While the agent is executing the solution sequence it ignores its percepts when choosing an action because it knows in advance what they will be.
  - An agent that carries out its plans with its eyes closed, so to speak. must be quite certain of what is going on.
  - Control theorists call this an open-loop system, because ignoring the percepts breaks the loop between agent and environment

- Well-defined problems and solutions
- A problem can be defined formally by five components:
  - INITIAL STATE: The initial state that the agent starts in_ For example, the initial state for our agent in Romania might be described as In(Arad).
  - ACTIONS: A description of the possible actions available to the agent Given a particular state s, ACTIONS(s) returns the set of actions that can be executed in s.
    - We say that each of these actions is applicable in s.
    - For example, from the state In(Arad), the applicable actions are { Go(Sibiu), Go(Timisoara), Go(Zerind)}.

- Transition Model: A description of what each action does; the formal name for this is the transition model, specified by a function RESULT(s, a) that returns the state that results from doing action a in state s.
  - We also use the term successor to refer to any state reachable from a given state by a single action.2 For example, we have RESULT(In(Arad), Go(Zerind)) = In(Zerind) .
- Together, the initial state, actions, and transition model implicitly define the state space of the problem—the set of all states reachable from the initial state by any sequence of actions.
  - The state space forms a directed network or graph in which the nodes are states and the links between nodes are actions.
- A path in the slate space is a sequence of states connected by a sequence of actions.

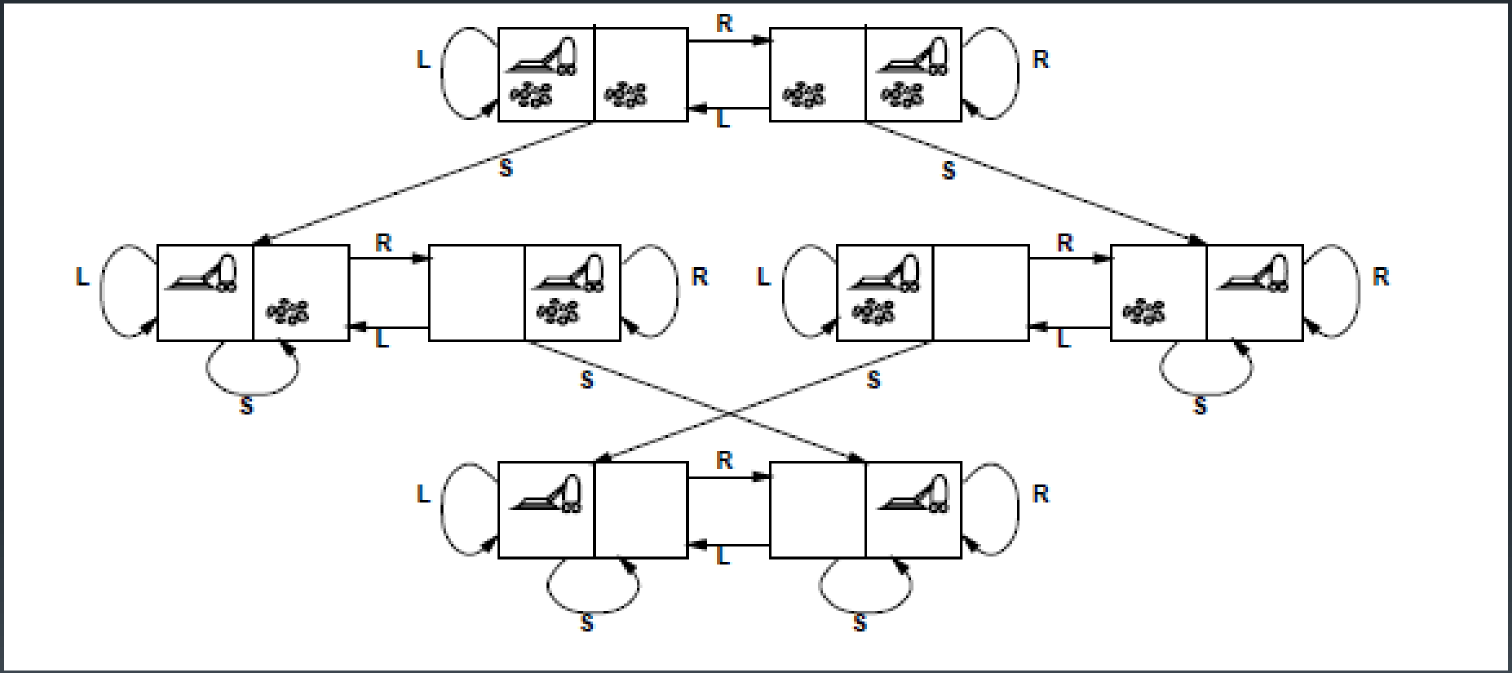
- Goal test: The goal test, which determines whether a given state is a goal state. Sometimes there is an explicit set of possible goal states, and the test simply checks whether the given state is one of them. The agent's goal in Romania is the singleton set { In(Bucharest)}.
  - Sometimes the goal is specified by an abstract property rather than an explicitly enumerated set of states. For example, in chess, the goal is to reach a state called "checkmate," where the opponent's king is under attack and can't escape.

- Path cost A path cost function that assigns a numeric cost to each path. The problem-solving agent chooses a cost function that reflects its own performance measure.
  - For the agent trying to get to Bucharest, time is of the essence, so the cost of a path might be its length in kilometers.
  - The step cost of taking action a in state s to reach state s' is denoted by e(s, a, s').
- The preceding elements define a problem and can be gathered into a single data structure that is given as input to a problem-solving algorithm.
- A solution to a problem is an action sequence that leads from the initial state to a goal state. Solution quality is measured by the path cost function, and an optimal solution has the lowest path cost among all solutions.

Selecting a state space

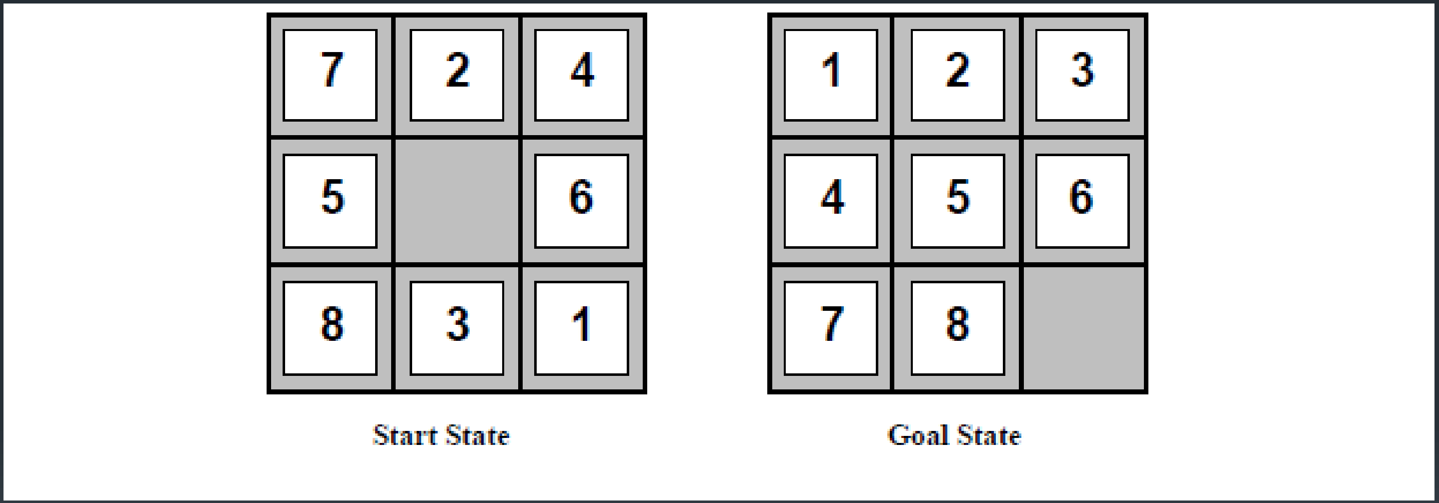
- Real world is absurdly complex
- State space must be abstracted for problem solving
- (Abstract) state = set of real states
- (Abstract) action = complex combination of real actions
  - e.g., \Arad ! Zerind" represents a complex set of possible routes, detours, rest stops, etc.
- For guaranteed realizability, any real state “in Arad“ must get to some real state “in Zerind”
- (Abstract) solution = set of real paths that are solutions in the real world
- Each abstract action should be easier" than the original problem!

- Example: vacuum world state space graph



- States: The state is determined by both the agent location and the dirt locations. The agent is in one of two locations, each of which might or might not contain dirt. Thus, there are 2×2^2 = B possible world states. A larger environment with n locations has $n * 2^n$ states.
- Initial state: Any state can be designated as the initial state.
- Actions: In this simple environment, each state has just three actions: Left, Right, and Suck. Larger environments might also include Up and Down.
- Transition model: The actions have their expected effects, except that moving Left in the leftmost square, moving Right in the rightmost square, and Sucking in a clean square have no effect.
- Goal test: This checks whether all the squares are clean.
- Path cost: Each step costs 1, so the path cost is the number of steps in the path.

- Example: The 8-puzzle



- States: A state description specifies the location of each of the eight Ides and the blank in one of the nine squares.
- Initial state: Any state can be designated as the initial state.
- Actions: The simplest formulation defines the actions as movements of the blank space Left, Right, Up, or Down. Different subsets of these are possible depending on where the blank is.
- Transition model: Given a state and action, this returns the resulting state.
- Goal test: This checks whether the state matches the goal configuration (Other goal configurations are possible.)
- Path cost: Each step costs 1, so the path cost is the number of steps in the path.

- Example: 8-queens problem

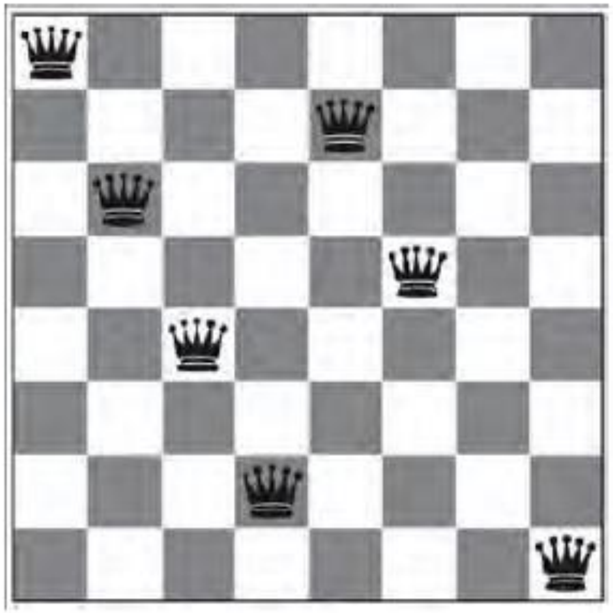


Figure 3.5 Almost a solution to the 8 queens problem. (Solution is left as an exercise.)

- There are two main kinds of formulation.
  - An incremental formulation involves operators that augment the state description, starting with an empty state; for the 8-queens problem, this means that each action adds a queen to the state.
  - A complete-state formulation starts with all 8 queens on the board and moves them amend. In either case, the path cost is of no interest because only the final state counts.

- With first incremental formulation
  - States: Any arrangement of 0 to 8 queens on the board is a state.
  - Initial state: No queens on the board.
  - Actions: Add a queen to any empty square.
  - Transition model: Returns the board with a queen added to the specified square.
  - Goal test: 8 queens are on the board, none attacked.

- Example: Airline travel problems
  - States: Each state obviously includes a location (e.g., an airport) and the current time. Furthermore, because the cost of an action (a flight segment) may depend on previous segments, their fare bases, and their status as domestic or international, the state must record extra information about these "historical" aspects.
  - Initial state: This is specified by the user's query.
  - Actions: Take any flight from the current location, in any seat class, leaving after the current time, leaving enough time for within-airport transfer if needed.
  - Transition model: The state resulting from taking a flight will have the flight's destination as the current location and the flight's arrival time as the current time.
  - Goal test: Are we at the final destination specified by the user?
  - Path cost: This depends on monetary cost, waiting time, flight time, customs and immigration procedures, seat quality, time of day, type of airplane, frequent-flyer mileage awards, and so on.

- Example: Traveling salesperson problem (TSP)
  - The traveling salesperson problem (TSP) is a touring problem in which each city must be visited exactly once.
  - The aim is to find the shortest tour.
  - The problem is known to be NP-hard, but an enormous amount of effort has been expended to improve the capabilities of TSP algorithms.
  - In addition to planning trips for traveling salespersons, these algorithms have been used for tasks such as planning movements of automatic circuit-board drills and of stocking machines on shop floors.

SEARCHING FOR SOLUTIONS

- A solution is an action sequence, so search algorithms work by considering various possible action sequences.
- The possible action sequences starting at the initial state form a search tree with the initial state at the root; the branches are actions and the nodes correspond to states in the state space of the problem.
- The root node of the tree corresponds to the initial state, In(Arad). The first step is to test whether this is a goal state.

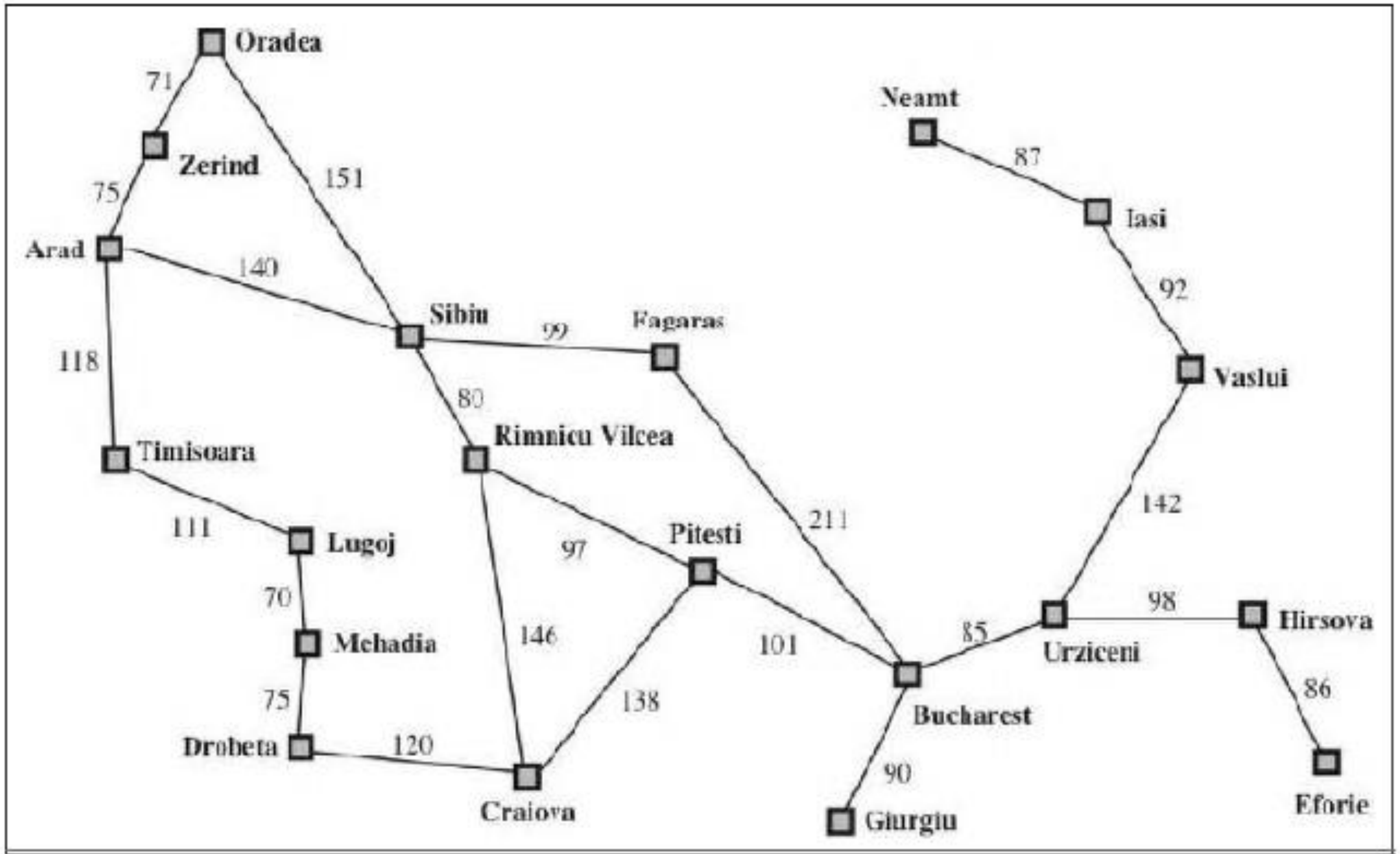


Figure 3.2 A simplified road map of part of Romania.

- Then we need to consider taking various actions. We do this by expanding the current state; that is, applying each legal action to the current state thereby generating a new set of states.
- In this case, we add three branches from the parent node In(Arad) leading to three new child nodes: in(Sibiu), In(Timisoara), and In(Zerind). Now we must choose which of these three possibilities to consider farther.

- Leaf node: A node with no children in the tree.
- The set of all leaf nodes available for expansion at any given point is called the frontier.
- The process of expanding nodes on the frontier continues until either a solution is found or there are no more states to expand.
- Search algorithms all share this basic structure; they vary primarily according to how they choose which state to expand next—the so-called search strategy.

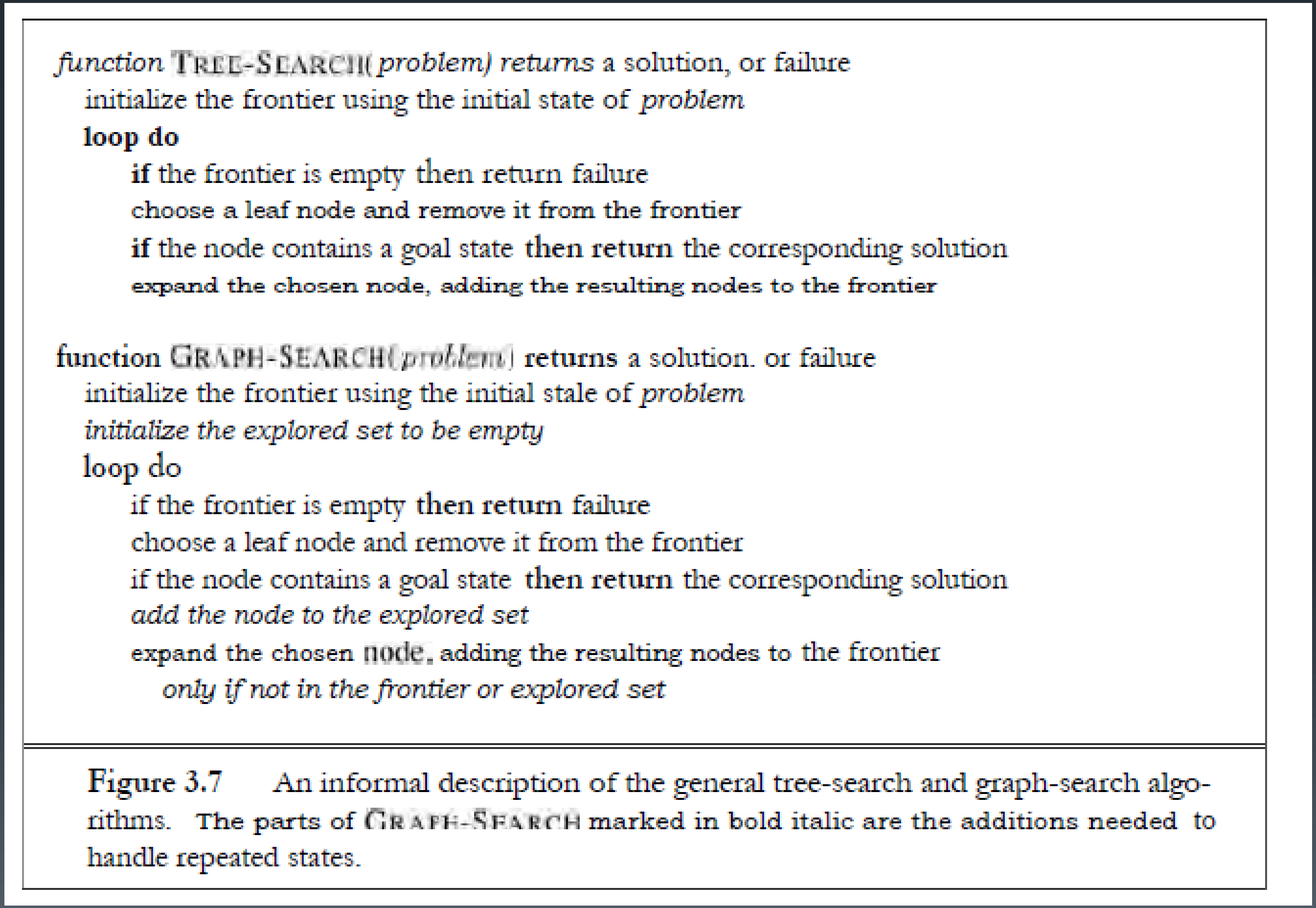
- General Tree search algorithms
- Basic idea:
  - offline, simulated exploration of state space
  - by generating successors of already-explored states (a.k.a. expanding states)

```
function Tree-Search( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
            if there are no candidates for expansion then return failure
            choose a leaf node for expansion according to strategy
            if the node contains a goal state then return the corresponding solution
            else expand the node and add the resulting nodes to the search tree
    end
```

Loopy Paths

- Repeated state in the search tree generated by a loopy path
- Loopy paths are a special case of the more general concept of redundant paths, which exist whenever there is more than one way to get from one state to another.
- loops can cause certain algorithms to fail, making otherwise solvable problems unsolvable.
- Fortunately, there is no need to consider loopy paths.
- We can rely on more than intuition for this: because path costs are additive and step costs are nonnegative, a loopy path to any given state is never better than the same path with the loop removed.

- The way to avoid exploring redundant paths is to remember where one has been.
- To do this, we set augment the TREE-SEARCH algorithm with a data structure called the explored set (also
- known as the closed list), which remembers every expanded node.
- Newly generated nodes that match previously generated nodes—ones in the explored set or the frontier—can be dis-carded instead of being added to the frontier. The new algorithm, called GRAPH-SEARCH



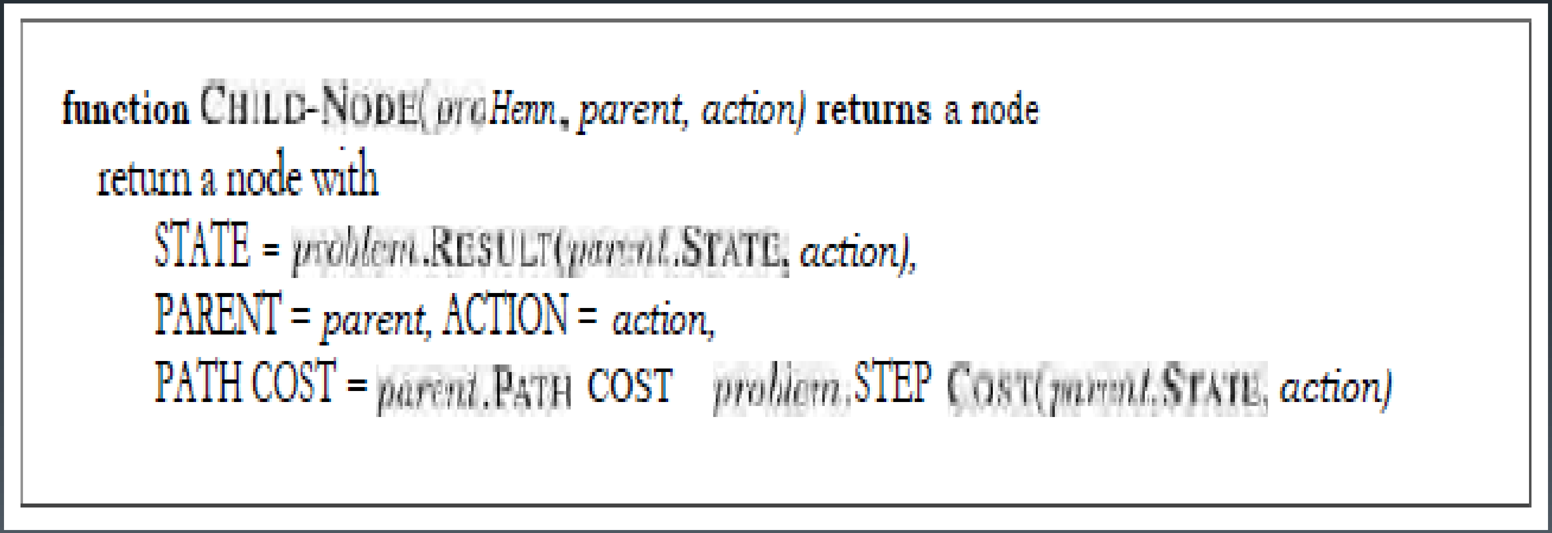
```
function TREE-SEARCH(problem) returns a solution, or failure
  initialize the frontier using the initial state of problem
  loop do
      if the frontier is empty then return failure
      choose a leaf node and remove it from the frontier
      if the node contains a goal state then return the corresponding solution
      expand the chosen node, adding the resulting nodes to the frontier

function GRAPH-SEARCH(problem) returns a solution. or failure
  initialize the frontier using the initial stale of problem
  initialize the explored set to be empty
  loop do
      if the frontier is empty then return failure
      choose a leaf node and remove it from the frontier
      if the node contains a goal state then return the corresponding solution
      add the node to the explored set
      expand the chosen node, adding the resulting nodes to the frontier
         only if not in the frontier or explored set
```

Figure 3.7 An informal description of the general tree-search and graph-search algorithms. The parts of GRAPH-SEARCH marked in bold italic are the additions needed to handle repeated states.

- Infrastructure for search algorithms
  - Search algorithms require a data structure to keep track of the search tree that is being constructed. For each node of the tree, we have a structure that contains four components:
    - STATE: the state in the state space to which the node corresponds;
    - PARENT: the node in the search tree that generated this node;
    - ACTION: the action that was applied to the parent to generate the node;
    - PATH-COST: the cost, traditionally denoted by g(n), of the path form the initial state to the node, as indicated by the parent pointers.
    - DEPTH: The number of steps along the path from the initial state.

- Given the components for a parent node, it is easy to see how to compute the necessary components for a child node.
- The function CHILD-NODE takes a parent node and an action and returns the resulting child node:



```
function CHILD-NODE(problem, parent, action) returns a node
    return a node with
        STATE = problem.RESULT(parent.STATE, action),
        PARENT = parent, ACTION = action,
        PATH COST = parent.PATH COST   problem.STEP COST(parent.STATE, action)
```

- Data Structure - Queue
  - The frontier needs to be stored in such a way that the search algorithm can easily choose the next node to expand according to its preferred strategy.
  - The appropriate data structure for this is a queue. The operations on a queue are as follows:
    - MAKE-QUEUE: creates queue with given elements
    - EMPTY(queue) returns true only if there are no more elements in the queue.
    - FIRST(queue) returns the first element of the queue.
    - REMOVE-FIRST(queue) returns the first element of the queue and removes it from the queue.
    - INSERT(element, queue) inserts an element and returns the resulting queue.
    - INSERT-ALL(elements, queue) inserts all element into the queue and returns the resulting queue.

- Measuring problem-solving performance
  - We can evaluate an algorithm's performance in four ways:
    - Completeness: Is the algorithm guaranteed to find a solution when there is one?
    - Optimality: Does the strategy find the optimal solution?
    - Time complexity: How long does it take to find a solution?
    - Space complexity: How much memory is needed to perform the search?

- In theoretical computer science, the typical measure is the size of the state space graph,; This is appropriate when the graph is an explicit data structure that is input to the search program.
- In AI, the graph is often represented implicitly by the initial state, actions, and transition model and is frequently infinite.
- For these reasons, complexity is expressed in terms of three quantities:
  - b, the branching factor or maximum number of successors of any node;
  - d. the depth of the shallowest goal node (i.e., the number of steps along the path from the root);
  - and m, the maximum length of any path in the state space.

- Time is often measured in terms of the number of nodes generated during the search, and space in terms of the maximum number of nodes stored in memory.
- To assess the effectiveness of a search algorithm, we can consider just the search cost— which typically depends on the time complexity but can also include a term for memory usage—or we can use the total cost, which combines the search cost and the path cost of the solution found.

Uninformed search strategies

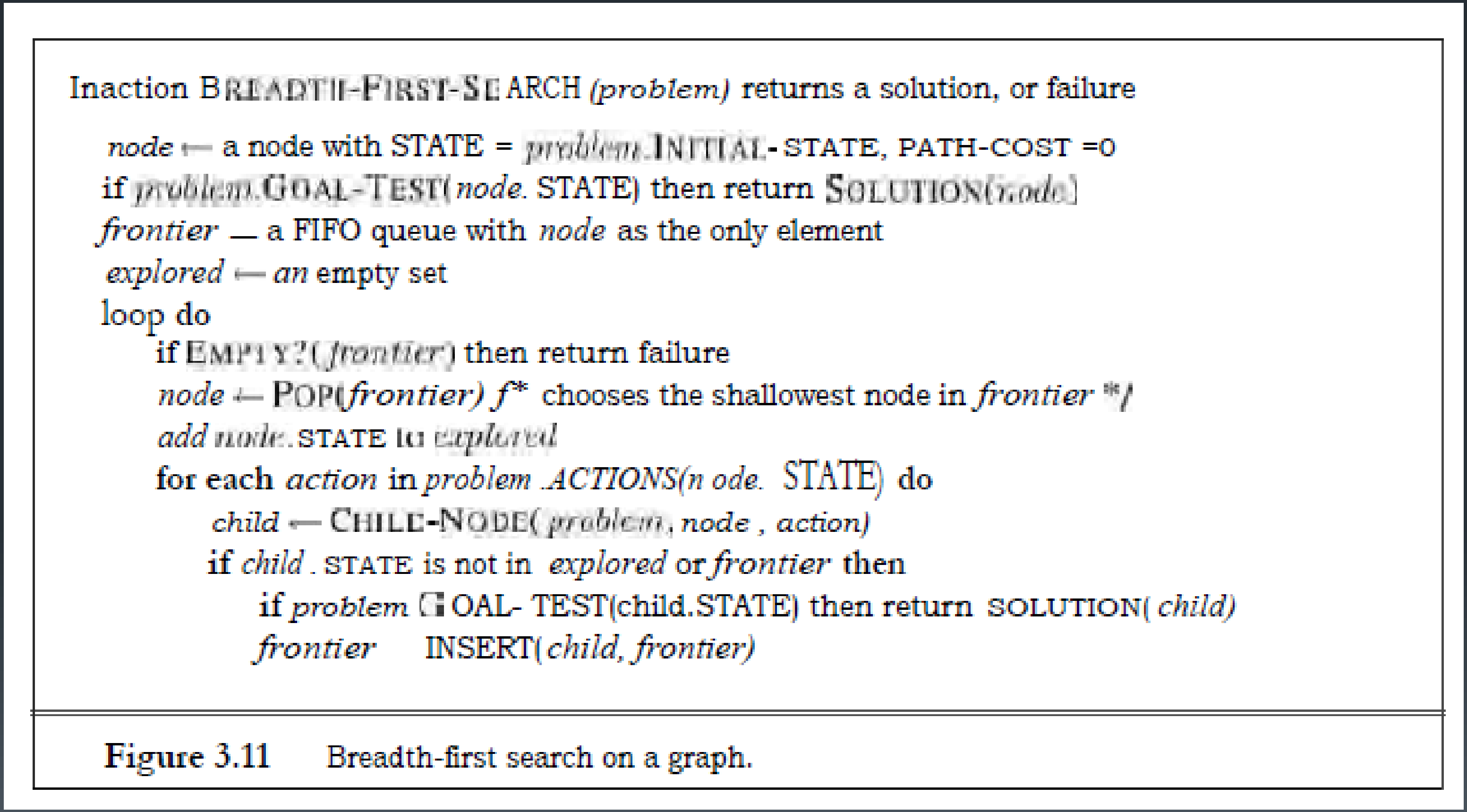
Uninformed search strategies

- The term means that the strategies have no additional information about states beyond that provided in the problem definition.
- All they can do is generate successors and distinguish a goal state from a non-goal state.
- All search strategies are distinguished by the order in which nodes are expanded.
- Strategies that know whether one non-goal state is "more promising" than another arc called informed search or heuristic search strategies

Breadth-First Search

- Is a simple strategy in which the root node is expanded first, then all the successors of the root node are expanded next, then their successors, and so on.
- In general, all the nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded.
- An instance of the general graph-search algorithm in which the shallowest unexpanded node is chosen for expansion.

- This is achieved very simply by using a FIFO queue for the frontier. Thus, new nodes (which are always deeper than their parents) go to the back of the queue, and old nodes, which are shallower than the new nodes, get expanded first.
- Breadth-first search is optimal if the path cost is a non-decreasing function of the depth of the node. The most common such scenario is that all actions have the same cost.
- The time and memory requirements are a problem for breadth first search.



```
Inaction BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
     if EMPTY?(frontier) then return failure
     node ← POP(frontier) /* chooses the shallowest node in frontier */
     add node.STATE to explored
     for each action in problem.ACTIONS(node.STATE) do
        child ← CHILD-NODE(problem, node, action)
        if child.STATE is not in explored or frontier then
           if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
           frontier    INSERT(child, frontier)
```

Figure 3.11 Breadth-first search on a graph.

Uniform-cost search

- Uniform-cost search expands the node n with the lowest path cost.
- This is done by storing the frontier as a priority queue ordered by q.
- It does not care about the number of steps a path has, but only of their total cost.
- There are two other significant differences from breadth-first search.
- May get stuck into infinite loop if it ever expands a node that a zero-cost action.

- Completeness is guaranteed if the cost of every step is greater or equal to some small positive constant Ɛ; this also ensures optimality.
- if C* be the cost of optimal solution and every action cost at least Ɛ, then the algorithms worst time and space complexity is O(b [C*/]) where 'b' is number of nodes at first level and 'd' depth
- If all step costs are equal then O(b [C*/]) equals O(bd)