Artificial Intelligence

Unit I

Introduction: What is AI?

Intelligent Agents: Agents and environment, Rationality, the nature of environment, the structure of agents.

Problem- solving by search: Problem-solving agents, Example problems, searching for solution, **uniformed** search strategies **informed** search strategies, Heuristic functions, On-line search agents and unknown environments.

.

What is Artificial Intelligence?

- Smart programs? Not really. Studying what is possible and underlying theories are very important.
    - E.g. How does a slow, tiny brain (biological or electrical) perceives, understands, and manipulates a complex world?
- Currently encompasses a large variety of subfields,
    - from general areas such as perception and logical reasoning to
    - specific tasks such as playing chess, writing poetry…
    - bringing together philosophy, logic, computer science, cognitive science and cognitive neuroscience

What is AI?

So defining artificial intelligence (AI) is hard.

In general, artificial intelligence is the field of science devoted to making computers perceive, reason, and act in ways that have, until now, been reserved for human beings.

The definitions concerned with thought processes and reasoning, ,behavior, means measure success in terms of fidelity to human performance,

Whereas - measure against an ideal performance measure, called rationality. A system is rational if it does the “right thing,” given what it knows.

Thinking Humanly	**Thinking Rationally**
"The exciting new effort to make computers think . . . *machines with minds*, in the full and literal sense." (Haugeland, 1985)	"The study of mental faculties through the use of computational models." (Charniak and McDermott, 1985)
"[The automation of] activities that we associate with human thinking, activities such as decision-making, problem solving, learning . . ." (Bellman, 1978)	"The study of the computations that make it possible to perceive, reason, and act." (Winston, 1992)
Acting Humanly	**Acting Rationally**
"The art of creating machines that perform functions that require intelligence when performed by people." (Kurzweil, 1990)	"Computational Intelligence is the study of the design of intelligent agents." (Poole *et al.*, 1998)
"The study of how to make computers do things at which, at the moment, people are better." (Rich and Knight, 1991)	"AI . . . is concerned with intelligent behavior in artifacts." (Nilsson, 1998)

Figure 1.1 Some definitions of artificial intelligence, organized into four categories.

Why AI?

- **Engineering:** To get machines to do a wider variety of useful things:

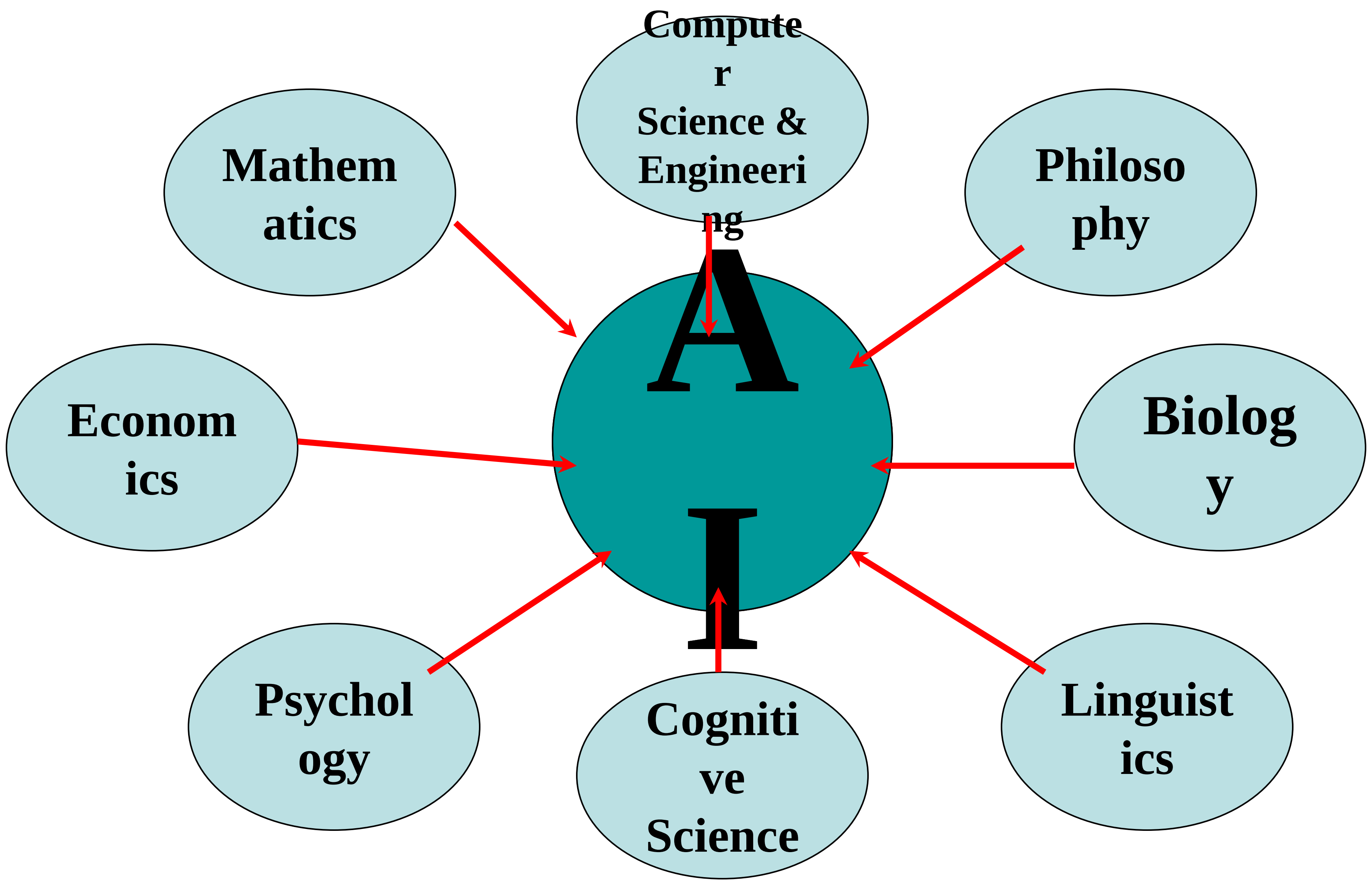
  e.g., find the best travel plan for your vacation, etc.
- **Cognitive Science:** As a way to understand how natural minds and mental phenomena work:

  e.g., visual perception, memory, learning, language, etc.
- **Philosophy:** As a way to explore some basic and interesting philosophical questions:

  e.g., the mind body problem, what is consciousness, etc.
- **Medicine:** As an automated doctor to know the patience disease:

  e.g., the blood pressure machine, and medical expert systems.

Foundations of AI



History of Artificial Intelligence

Stone age (1943-1956)

- Early work on neural networks and logic **(Herbert Simon,1943)**
- Birth of AI: Dartmouth workshop - summer 1956
- John McCarthy's name for the field: artificial intelligence

Herbert Simon in 1991

John McCarthy in AAAI-2002

Early enthusiasm, great expectations (1952-1969)

Dark ages (1969-1973)Why?

- AI did not scale up: combinatorial explanation.
- Failure of natural language translation approach based on simple grammars and word dictionary.
- Funding for natural language processing stopped.
- Failure of perceptrons to learn such a simple function.
- Realization of the difficulty of the learning process.
- There is no Symbolic concept learning.

Renaissance (1969-1979)

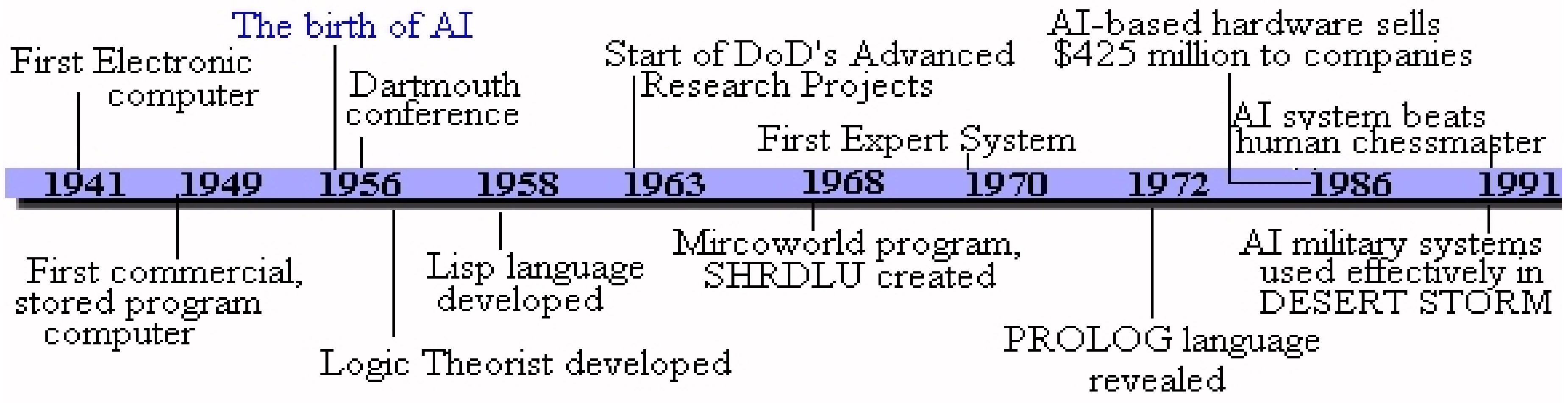
Change of problem solving paradigm: From search-based problem solving to knowledge-based problem solving.

Industrial age (1980-present)

- Building expert systems with formal knowledge acquisition techniques.
- There exists successful commercial expert systems.
- Many AI companies.
- Exploration of different learning strategies.
- Reasoning, Genetic algorithms, Neural networks, etc.

The Maturity and return of neural networks :1986-1995

- Many successful applications of neural networks , machine learning, and data mining.
- Base claims on theorems and experiments rather than on intuition.
- Show relevance to real-world applications rather than toy examples.
- The reinvention of the back propagation learning algorithm for neural networks and other optimization process.


The birth of AI
First Electronic computer
Dartmouth conference
Start of DoD's Advanced Research Projects
AI-based hardware sells $425 million to companies
AI system beats human chessmaster
First Expert System
1941
1949
1956
1958
1963
1968
1970
1972
1986
1991
First commercial, stored program computer
Logic Theorist developed
Lisp language developed
Mircoworld program, SHRDLU created
PROLOG language revealed
AI military systems used effectively in DESERT STORM

Intelligent agents (1995-present)

The realization that the previously isolated subfields of AI (speech recognition, planning, robotics, computer vision, machine learning, knowledge representation, etc.) need to be reorganized when their results are to be tied together into a single agent design.

A process of reintegration of different sub-areas of AI to build a “whole agent”:

- multi-agent systems;
- agents for different types of applications, web search engine agents.

See the Site:
http://www.stottlerhenke.com/ai_general/history.htm
for more details.

1997: The Deep Blue chess program beats the current world chess champion, Garry Kasparov, in a widely followed match.

First official Robo-Cup soccer match featuring table-top matches with 40 teams of interacting robots and over 5000 spectators.

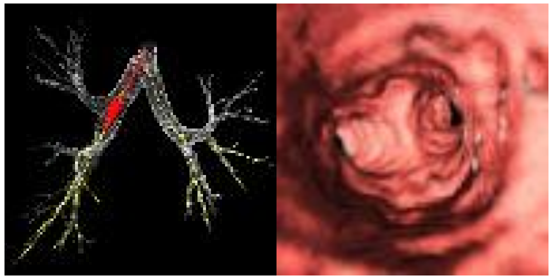
Late 90's: Web crawlers and other AI-based information extraction programs become essential in widespread use of the world-wide-web. Demonstration of an Intelligent Room and Emotional Agents at MIT's AI Lab. Initiation of work on the Oxygen Architecture, which connects mobile and stationary computers in an adaptive network.

2000: Interactive robot pets (a.k.a. "smart toys") become commercially available, realizing the vision of the 18th cen. novelty toy makers. Cynthia Breazeal at MIT publishes her dissertation on Sociable Machines, describing KISMET, a robot with a face that expresses emotions. The Nomad robot explores remote regions of Antarctica looking for meteorite samples.
Today*:* See AI *in the newsin the news:* *http://www.aaai.org/AITopics/html/current.html* for history *in the making*!

Some Achievements and State of the Art in AI

- Computers have won over world champions in several games, including Checkers, Othello, and Chess, but still do not do well in Go
- AI techniques are used in many systems and applications, e.g.: formal calculus, video games, route planning, logistics planning, pharmaceutical drug design, medical diagnosis, hardware and software trouble-shooting, speech recognition, road traffic monitoring, facial recognition, medical image analysis, part inspection, etc...
- Some industries (automobile, electronics) are highly robotized, while other robots perform brain and heart surgery, are rolling on Mars, fly autonomously, …, but home robots remain mostly a thing of the future

THE POWER OF T TECHNOLOGY
INTEGRATION OF TWO REVOLUTIONARY TECHNOLOGIES
FULL-BODY
100% Frameless Radiosurgery



Typical Applications of AI

- **Optical Character Recognition (OCR): Scanning typewritten/handwritten documents, finger prints, etc.**
- **Voice Recognition: Transcribing spoken words into ASCII text.**
- **Medical Diagnosis: Assisting doctors with their diagnosis by analyzing the reported symptoms and/or medical imaging data such as MRIs or X-rays.**
- **Oil Industry: To assist in predict PVT properties, Permeability Prediction, and Measure the reservoir Characterization,…etc.**
- **Target Recognition: Military application which uses video and/or infrared image data to determine if an enemy target is present.**
- **Targeted Marketing: Finding the set of demographics which have the highest response rate for a particular marketing campaign.**
- **Intelligent Searching: An internet search engine that provides the most relevant content and banner advertisements based on the users' past behavior.**
- **Fraud Detection: Detect fraudulent credit card transactions and automatically decline the charge.**

What can AI systems do?

Here are some example applications

- **Computer vision:** face recognition from a large set
- **Robotics:** autonomous (mostly) automobile
- **Natural language processing:** simple machine translation
- **Expert systems:** medical diagnosis in a narrow domain
- **Spoken language systems:** ~1000 word continuous speech
- **Planning and scheduling:** Hubble Telescope experiments
- **Learning:** text categorization into ~1000 topics
- **User modeling:** Bayesian reasoning in Windows help (the infamous paper clip…)
- **Games:** Grand Master level in chess (world champion), checkers, etc.

What can't AI systems do yet?

- Understand natural language robustly (e.g., read and understand articles in a newspaper)
- Interpret an arbitrary visual scene
- Learn a natural language
- Construct plans in dynamic real-time domains
- Refocus attention in complex environments
- Perform life-long learning

AI Common Programming Languages

Plankalkul

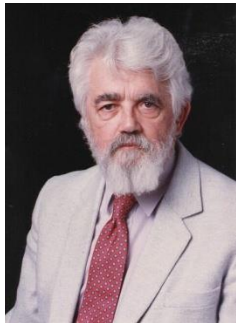
Konrad Zuse

Fortran

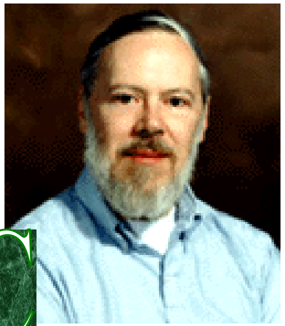
John Backus

Python

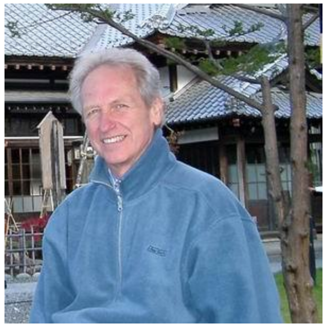
LISP



John McCarthy



C



Prolog

MATL
AB

C++

Java

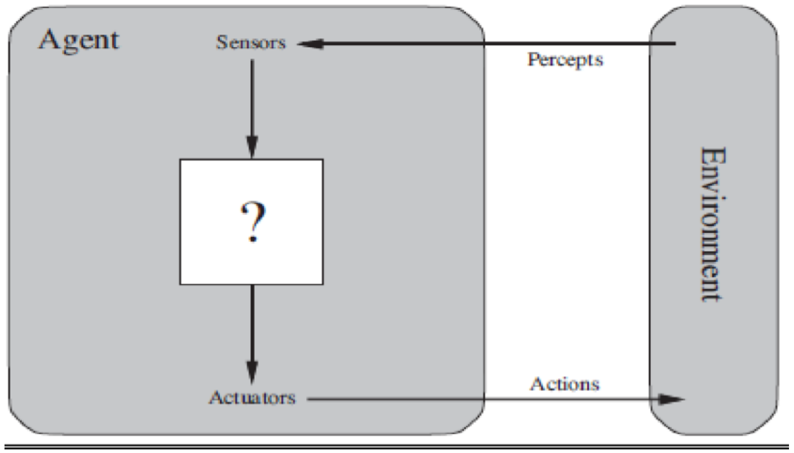
Intelligent Agent

- **An Intelligent Agent perceives its environment via sensors and acts rationally upon that environment with its effectors. Hence, an agent gets percepts one at a time, and maps this percept sequence to actions.**

agent = architecture + program

Human agent, Robotic agent, software agent

agent's behavior is described by the **agent function** that maps any given percept sequence to an action.



Agents interact with environments through sensors and actuators.

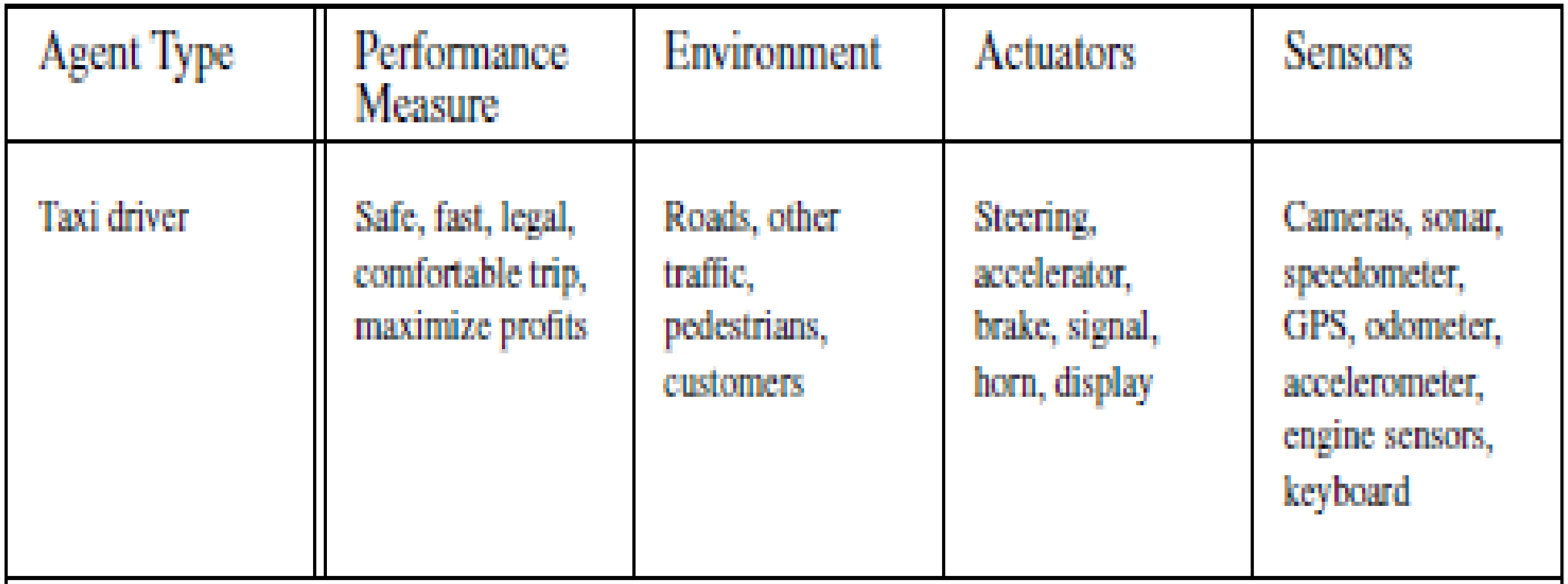
rational agent: *For each possible percept sequence, a rational agent should select an* ***action*** *that is expected to maximize its* ***performance measure****, given the* ***evidence*** *provided by the* ***percept sequence*** *and whatever built-in knowledge (****priori****) the agent has.*

Agent - Omniscience, learning, and autonomy

Rationality maximizes *expected* performance, while perfection maximizes *actual* performance.

Specifying the task environment

- **PEAS** (**P**erformance, **E**nvironment, **A**ctuators, **S**ensors)



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.

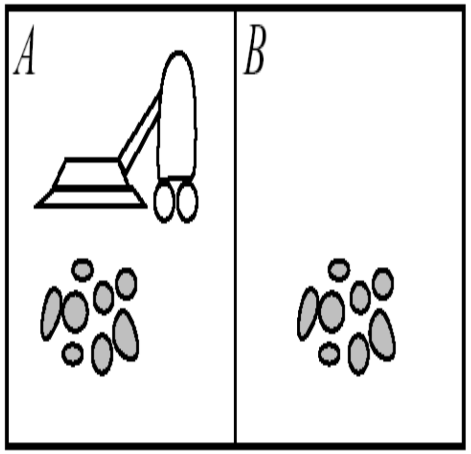
Artificial Agent Examples

Automated taxi driving system Example

- **Percepts**: Video, speedometer, odometer, engine sensors, keyboard input, microphone, …
- **Actions**: Steer, accelerate, brake, speak/display, …
- **Goals**: Maintain safety, reach destination, maximize profits (fuel, tire wear), obey laws, provide passenger comfort, …
- **Environment**: Dhahran streets, freeways, traffic, weather, customers, …, etc.
- **Different aspects of driving may require different types of agent programs!**

Another example:

Vacuum-cleaner world



Percepts: location and contents, e.g., $[A, Dirty]$

Actions: $Left$, $Right$, $Suck$, $NoOp$

Task Environment types

- Fully observable (vs. partially observable): An agent's sensors give it access to the complete state of the environment at each point in time.
- 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.
- the agent need not maintain any internal state to keep track of the world

Task Environment types

- 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,
- and an automated taxi cannot see what other drivers are thinking.

Task Environment types

- Single agent (vs. multiagent): An agent operating by itself in an environment.
- crossword puzzle - Single agent
- Chess – Competitive multiagent environment
- Taxi Driver - cooperative multiagent – to avoid collisions
- partially competitive – to occupy parking space

Task Environment types

- Multiagent to behave rational
- Communication
- in some competitive environments, **randomized behavior is rational because** it avoids the pitfalls of predictability.

Task Environment types

- Deterministic (vs. stochastic): The next state of the environment is completely determined by the current state and the action executed by the agent.
- Adv: an agent need not worry about uncertainty in a fully observable, deterministic environment.
- Eg: The vacuum world, Taxi driving

Task Environment types

- a **nondeterministic environment is one in which actions are** characterized by their *possible outcomes, but no probabilities are attached to them. Nondeterministic*
- environment descriptions are usually associated with performance measures that require the agent to succeed for *all possible outcomes of its actions.*

Task Environment types

- Episodic (vs. sequential): The agent's experience is divided into atomic "episodes" (each episode consists of the agent perceiving and then performing a single action), and the choice of action in each episode depends only on the episode itself.
- Eg: Many classification tasks are episodic.
- an agent that has to spot defective parts on an assembly line

Task Environment types

- In sequential environments the current decision could affect all future decisions.

Eg: Chess and taxi driving

- in both cases, short-term actions can have long-term consequences.

Environment types

- Static (vs. dynamic): The environment is unchanged while an agent is deliberating. (The environment is semidynamic if the environment itself does not change with the passage of time but the agent's performance score does)
- Crossword puzzles, Chess, when played with a clock, Taxi driving

Environment types

- Discrete (vs. continuous): A limited number of distinct, clearly defined percepts and actions.

Eg: chess Environment, Taxi Driving.

Environment types

- **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.
- **Different from fully /partially observable**

Eg: solitaire card games, new video game,

Environment types

	Chess with a clock	Chess without a clock	Taxi driving
Fully observable	Yes	Yes	No
Deterministic	Strategic	Strategic	No
Episodic	No	No	No
Static	Semi	Yes	No
Discrete	Yes	Yes	No
Single agent	No	No	No

- The environment type largely determines the agent design
- The real world is (of course) partially observable, stochastic, sequential, dynamic, continuous, multi-agent

Summary

- **Agent:** an entity that perceives and acts; or, one that can be viewed as perceiving and acting. Essentially any object qualifies; the key point is the way the object implements an agent function
- **Agent function:** a function that specifies the agent's action in response to every possible percept sequence.
- **Agent program:** that program which, combined with a machine architecture, implements an agent function. In our simple designs, the program takes a new percept on each invocation and returns an action.
- **Rationality:** a property of agents that choose actions that maximize their expected utility, given the percepts to date.
- **Autonomy:** a property of agents whose behavior is determined by their own experience rather than solely by their initial programming.

Quiz 1

Examine the AI literature to discover whether the following tasks can currently be solved by computers:

- a. Playing a decent game of table tennis (Ping-Pong).
- b. Driving in the center of Cairo, Egypt.
- c. Driving in Victorville, California.
- d. Buying a week's worth of groceries at the market.
- e. Buying a week's worth of groceries on the Web.
- f. Playing a decent game of bridge at a competitive level.

PEAS description of the task environment for:

- Interactive English tutor, Automated taxi Driver , Medical Diagnosis System, Playing soccer, Knitting a sweater and Bidding on an item at an auction.

Task environments and their characteristics for:

- Automated taxi Driver , Medical Diagnosis, part picking Robot, Interactive English tutor

For each of the following assertions, say whether it is true or false and support your answer with examples or counterexamples where appropriate.

a. An agent that senses only partial information about the state cannot be perfectly rational.

b. There exist task environments in which no pure reflex agent can behave rationally.

c. There exists a task environment in which every agent is rational.

To what extent (Limited, borderline, Not AI , complete AI) are the following computer systems instances of artificial intelligence: Justify with reasons.

• Supermarket bar code scanners.

• Web search engines.

• Voice-activated telephone menus.

• Internet routing algorithms that respond dynamically to the state of the network.

•Define in your own words:

AI, rationality thinking, logical reasoning ,Agent, Agent Function, Agent program, table driven agent, simple reflex agent, model based agent, utility based agent, learning agent, state space, search tree and successor function.

* Mention the difference between a world state, a state description, and a search node?

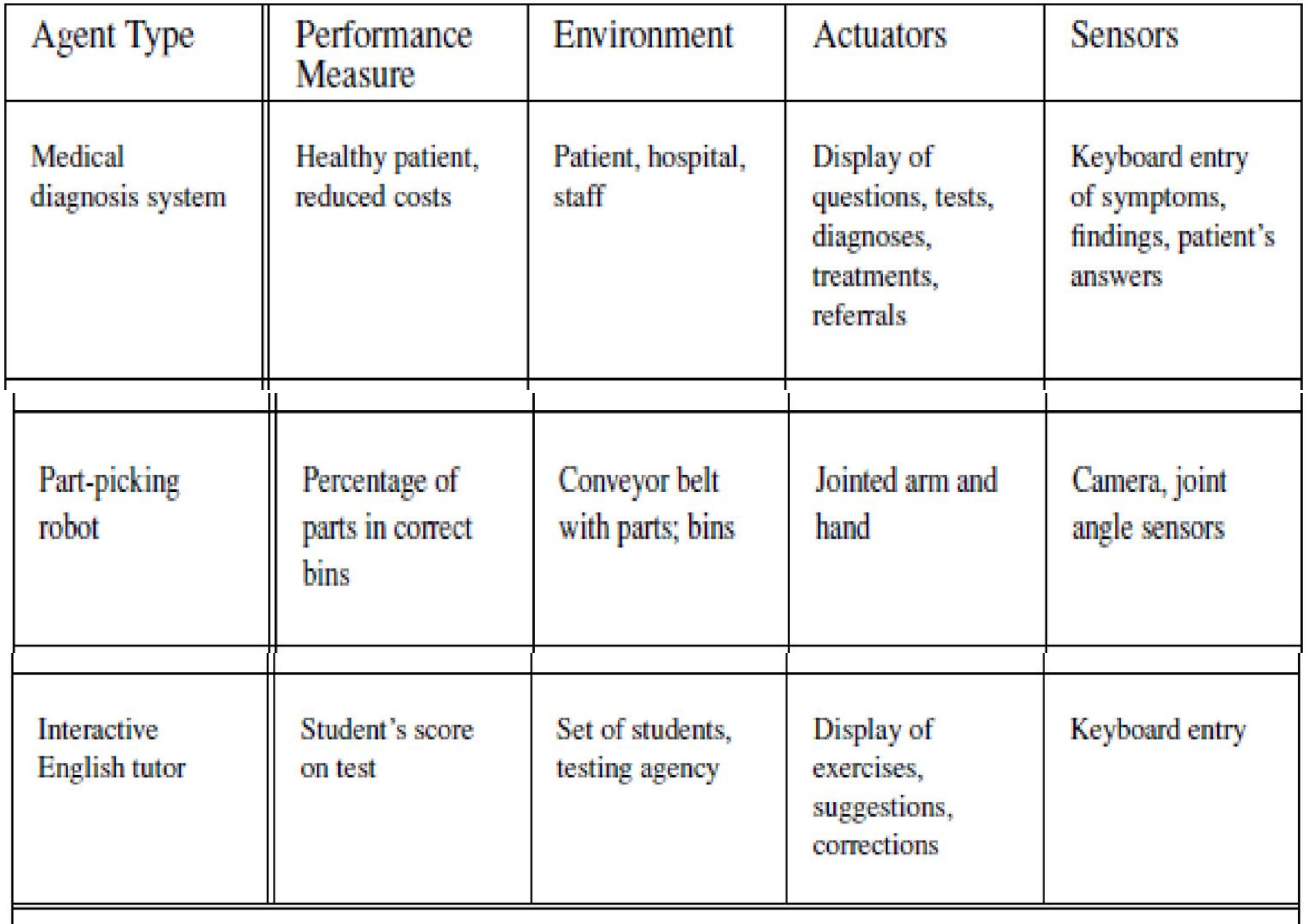
Soln.

- **(ping-pong)** A reasonable level of proficiency was achieved by Andersson's robot (Andersson, 1988).
- **(driving in Cairo)** No. Although there has been a lot of progress in automated driving, all such systems currently rely on certain relatively constant clues: that the road has shoulders and a center line, that the car ahead will travel a predictable course, that cars will keep to their side of the road, and so on. Some lane changes and turns can be made on clearly marked roads in light to moderate traffic. Driving in downtown Cairo is too unpredictable for any of these to work.
- **(driving in Victorville, California)** Yes, to some extent, as demonstrated in DARPA's Urban Challenge. Some of the vehicles managed to negotiate streets, intersections, well-behaved traffic, and well-behaved pedestrians in good visual conditions.

Soln. contd.

- **(shopping at the market)** No. No robot can currently put together the tasks of moving in a crowded environment, using vision to identify a wide variety of objects, and grasping the objects (including squishable vegetables) without damaging them. The component pieces are nearly able to handle the individual tasks, but it would take a major integration effort to put it all together.
- **(shopping on the web)** Yes. Software robots are capable of handling such tasks, particularly if the design of the web grocery shopping site does not change radically over time.
- **(bridge) Yes.** Programs such as GIB now play at a solid level.

Soln.



Agent Type	Performance Measure	Environment	Actuators	Sensors
Medical diagnosis system	Healthy patient, reduced costs	Patient, hospital, staff	Display of questions, tests, diagnoses, treatments, referrals	Keyboard entry of symptoms, findings, patient's answers
Part-picking robot	Percentage of parts in correct bins	Conveyor belt with parts; bins	Jointed arm and hand	Camera, joint angle sensors
Interactive English tutor	Student's score on test	Set of students, testing agency	Display of exercises, suggestions, corrections	Keyboard entry

Task Environment	Observable	Agents	Deterministic	Episodic	Static	Discrete
Taxi driving	Partially	Multi	Stochastic	Sequential	Dynamic	Continuous
Medical diagnosis	Partially	Single	Stochastic	Sequential	Dynamic	Continuous
Part-picking robot	Partially	Single	Stochastic	Episodic	Dynamic	Continuous
Interactive English tutor	Partially	Multi	Stochastic	Sequential	Dynamic	Discrete

•a.An agent that senses only partial information about the state cannot be perfectly rational.

•False. Perfect rationality refers to the ability to make good decisions given the sensor information received.

•b. There exist task environments in which no pure reflex agent can behave rationally.

•True. A pure reflex agent ignores previous percepts, so cannot obtain an optimal state estimate in a partially observable environment. For example, correspondence chess is played by sending moves; if the other player's move is the current percept, a reflex agent could not keep track of the board state and would have to respond to, say, "a4" in the same way regardless of the position in which it was played.

•c. There exists a task environment in which every agent is rational.

•True. For example, in an environment with a single state, such that all actions have the same reward, it doesn't matter which action is taken. More generally, any environment that is reward-invariant under permutation of the actions will satisfy this property.

•Although bar code scanning is in a sense computer vision, these are not AI systems. The problem of reading a bar code is an extremely limited and artificial form of visual interpretation, and it has been carefully designed to be as simple as possible, given the hardware.

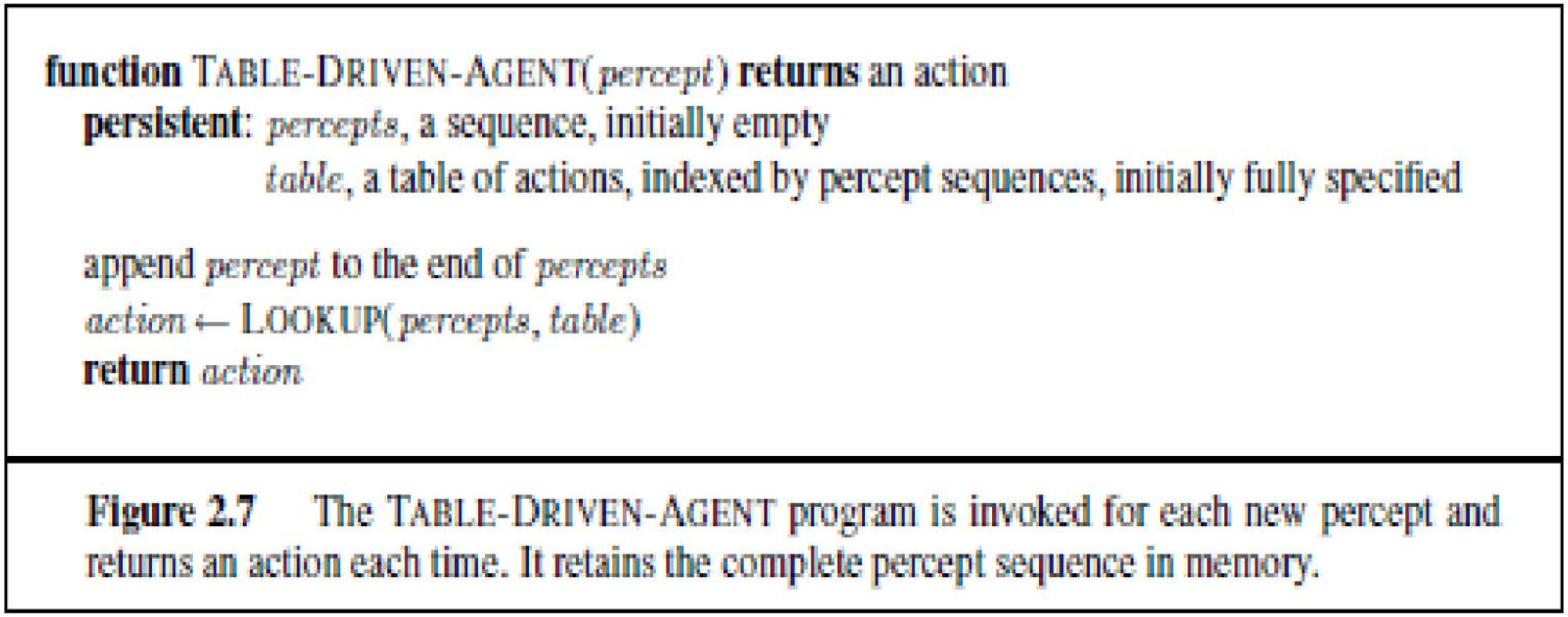
•In many respects. The problem of determining the relevance of a web page to a query is a problem in natural language understanding, Search engines like Ask.com, which group the retrieved pages into categories, use clustering techniques. Likewise, other functionalities provided by a search engines use intelligent techniques; for instance, the spelling corrector uses a form of data mining based on observing users' corrections of their own spelling errors. On the other hand, the problem of indexing billions of web pages in a way that allows retrieval in seconds is a problem in database design, not in artificial intelligence.

•To a limited extent. Such menus tends to use vocabularies which are very limited –e.g. the digits, "Yes", and "No" — and within the designers' control, which greatly simplifies the problem. On the other hand, the programs must deal with an uncontrolled space of all kinds of voices and accents. The voice activated directory assistance programs used by telephone companies, which must deal with a large and changing vocabulary are certainly AI programs.

•This is borderline. There is something to be said for viewing these as intelligent agents working in cyberspace. The task is sophisticated, the information available is partial, the techniques are heuristic (not guaranteed optimal), and the state of the world is dynamic.All of these are characteristic of intelligent activities. On the other hand, the task is very far from those normally carried out in human cognition.

THE STRUCTURE OF AGENTS

- *agent = architecture + program*
- *Agent programs:* **Table-driven agents**
  - use a percept sequence/ action table in memory to find the next action. They are implemented by a (large) lookup table



```
function TABLE-DRIVEN-AGENT(percept) returns an action
   persistent: percepts, a sequence, initially empty
               table, a table of actions, indexed by percept sequences, initially fully specified

   append percept to the end of percepts
   action ← LOOKUP(percepts, table)
   return action
```

Figure 2.7 The TABLE-DRIVEN-AGENT program is invoked for each new percept and returns an action each time. It retains the complete percept sequence in memory.

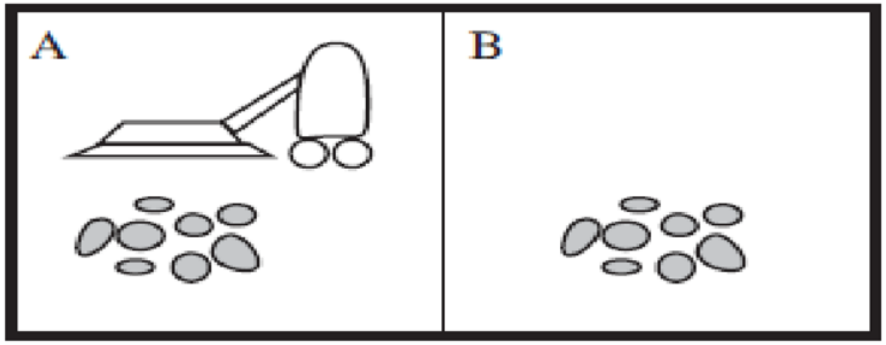


Figure 2.2 A vacuum-cleaner world with just two locations.

Percept sequence	Action
$[A, Clean]$	$Right$
$[A, Dirty]$	$Suck$
$[B, Clean]$	$Left$
$[B, Dirty]$	$Suck$
$[A, Clean], [A, Clean]$	$Right$
$[A, Clean], [A, Dirty]$	$Suck$
$\vdots$	$\vdots$
$[A, Clean], [A, Clean], [A, Clean]$	$Right$
$[A, Clean], [A, Clean], [A, Dirty]$	$Suck$
$\vdots$	$\vdots$

Figure 2.3 Partial tabulation of a simple agent function for the vacuum-cleaner world shown in Figure 2.2.

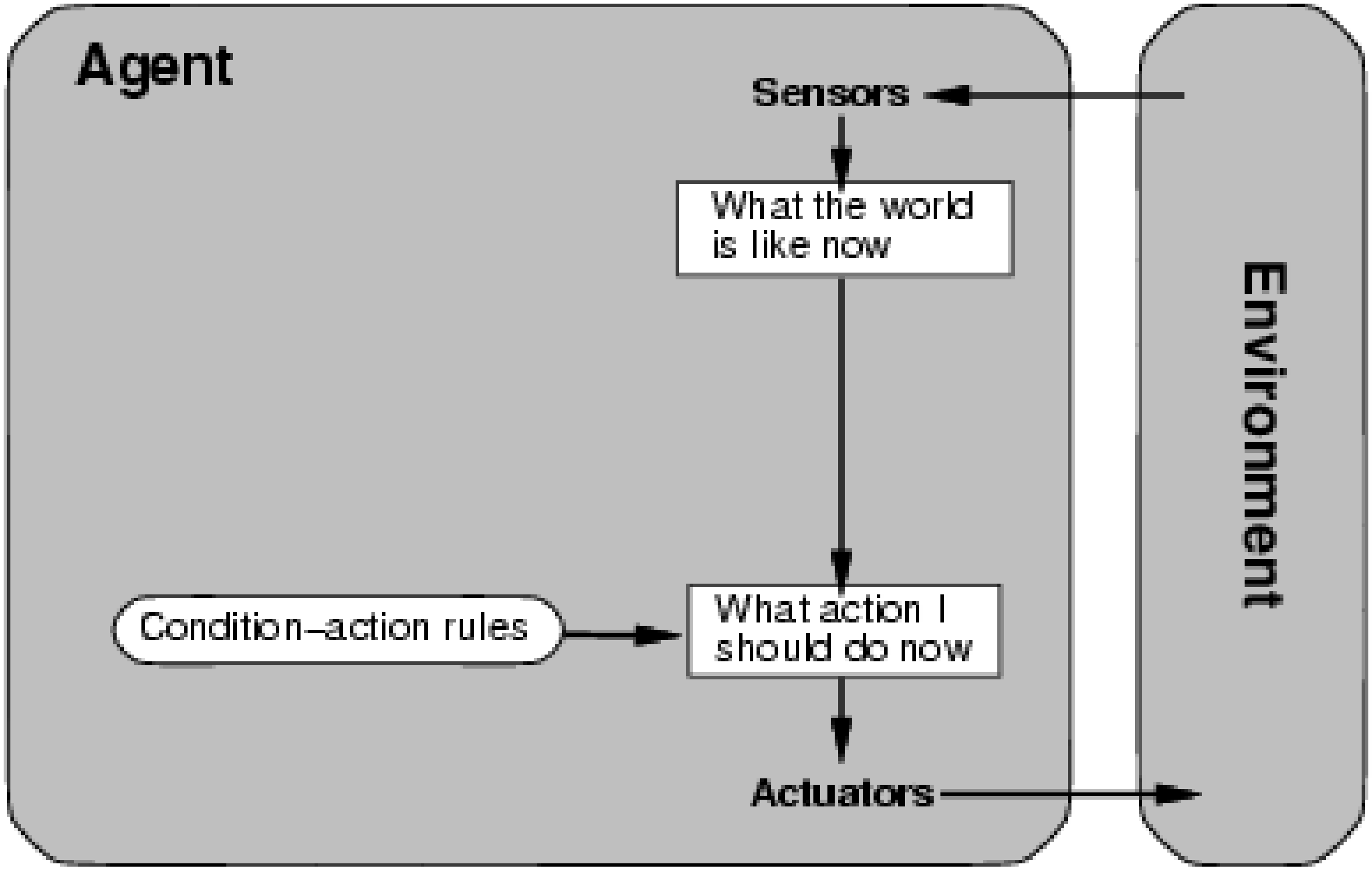
Table-lookup agent

- Drawbacks:
  - Huge table
  - Take a long time to build the table
  - No autonomy
  - Even with learning, need a long time to learn the table entries

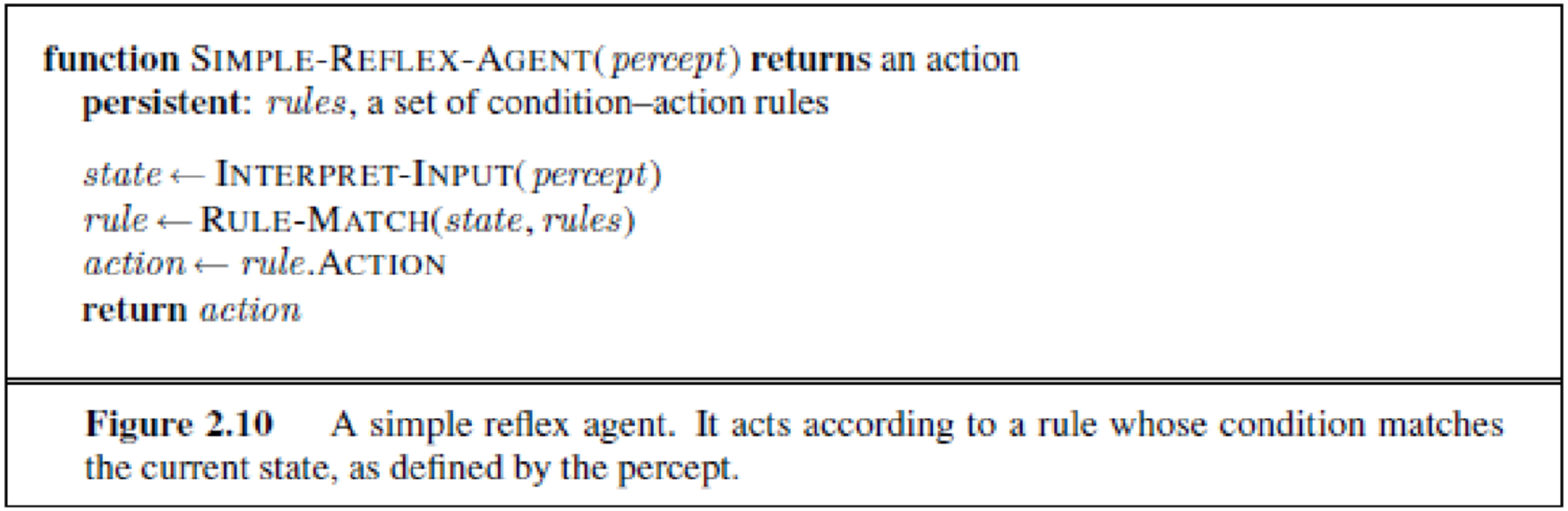
Agent Program types

- Simple reflex agents
- Model-based reflex agents
- Goal-based agents
- Utility-based agents

Simple reflex agents

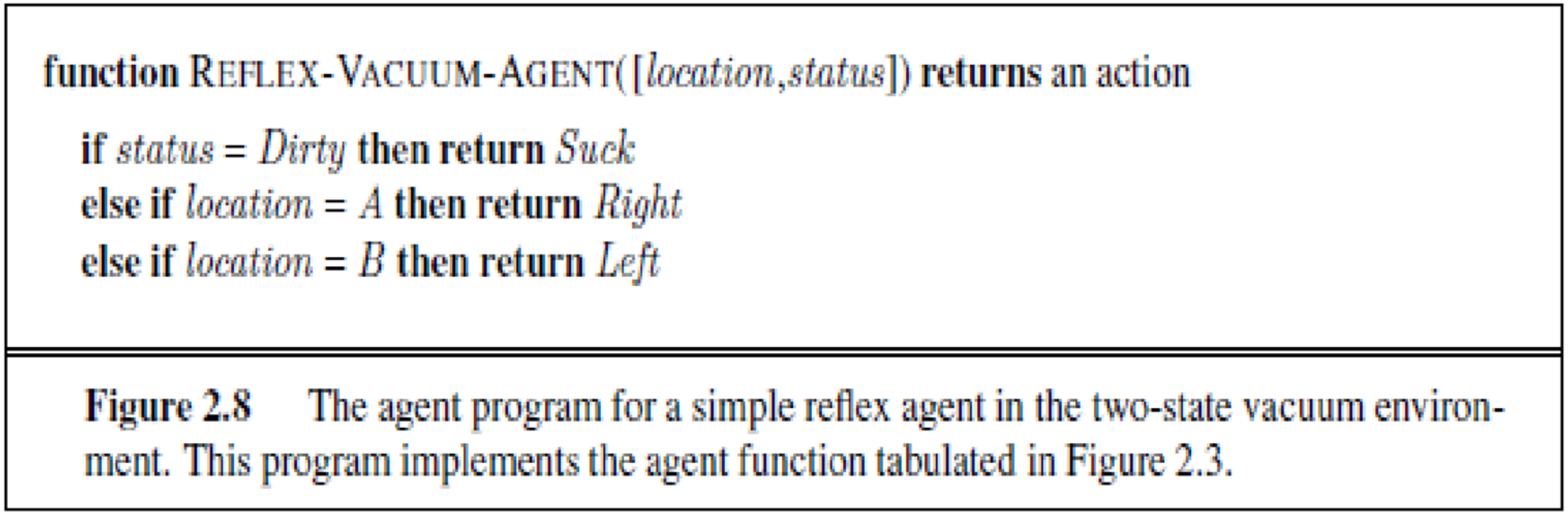


Simple reflex agents



```
function SIMPLE-REFLEX-AGENT(percept) returns an action
  persistent: rules, a set of condition–action rules

  state ← INTERPRET-INPUT(percept)
  rule ← RULE-MATCH(state, rules)
  action ← rule.ACTION
  return action
```

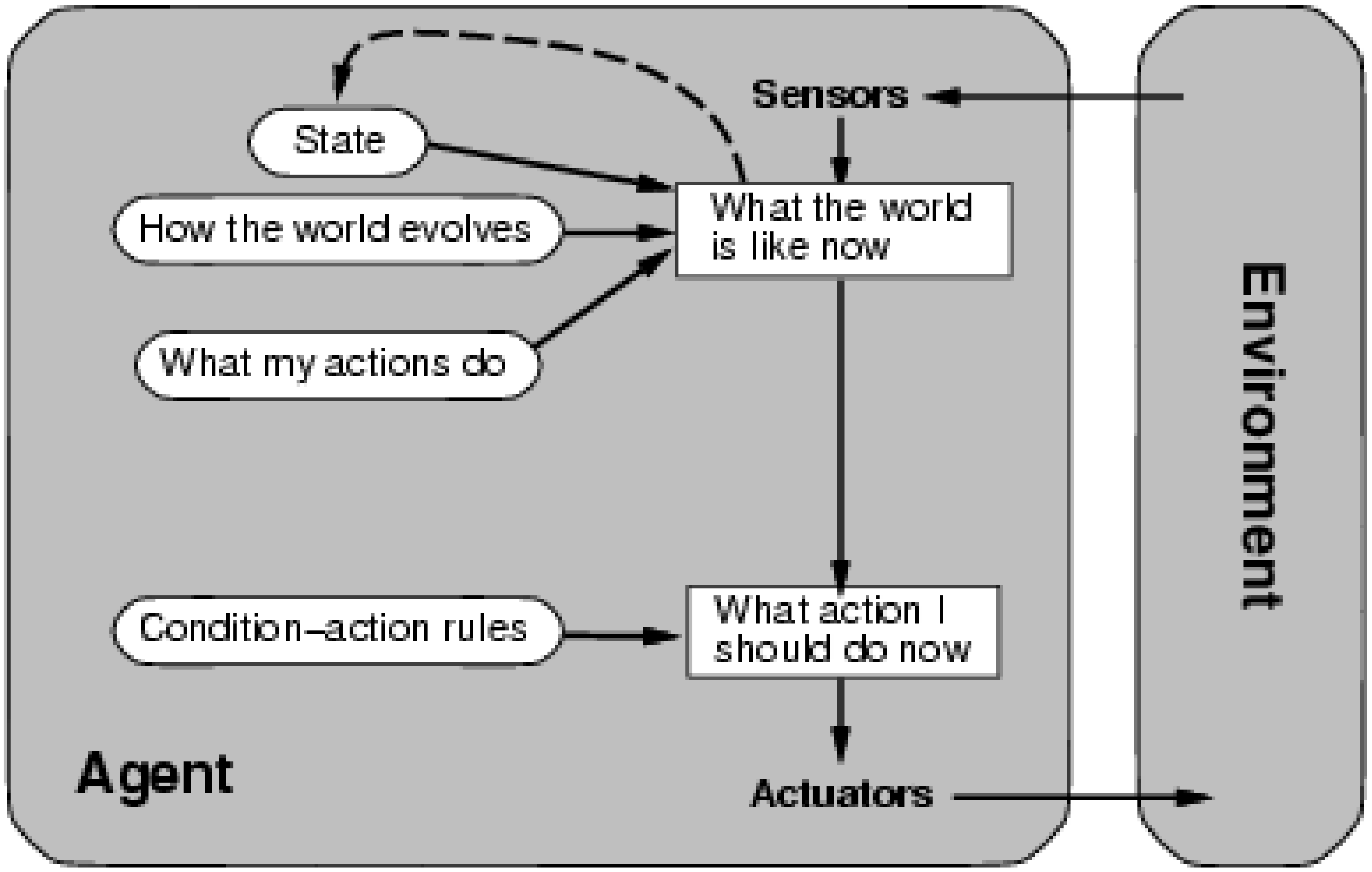
Figure 2.10 A simple reflex agent. It acts according to a rule whose condition matches the current state, as defined by the percept.



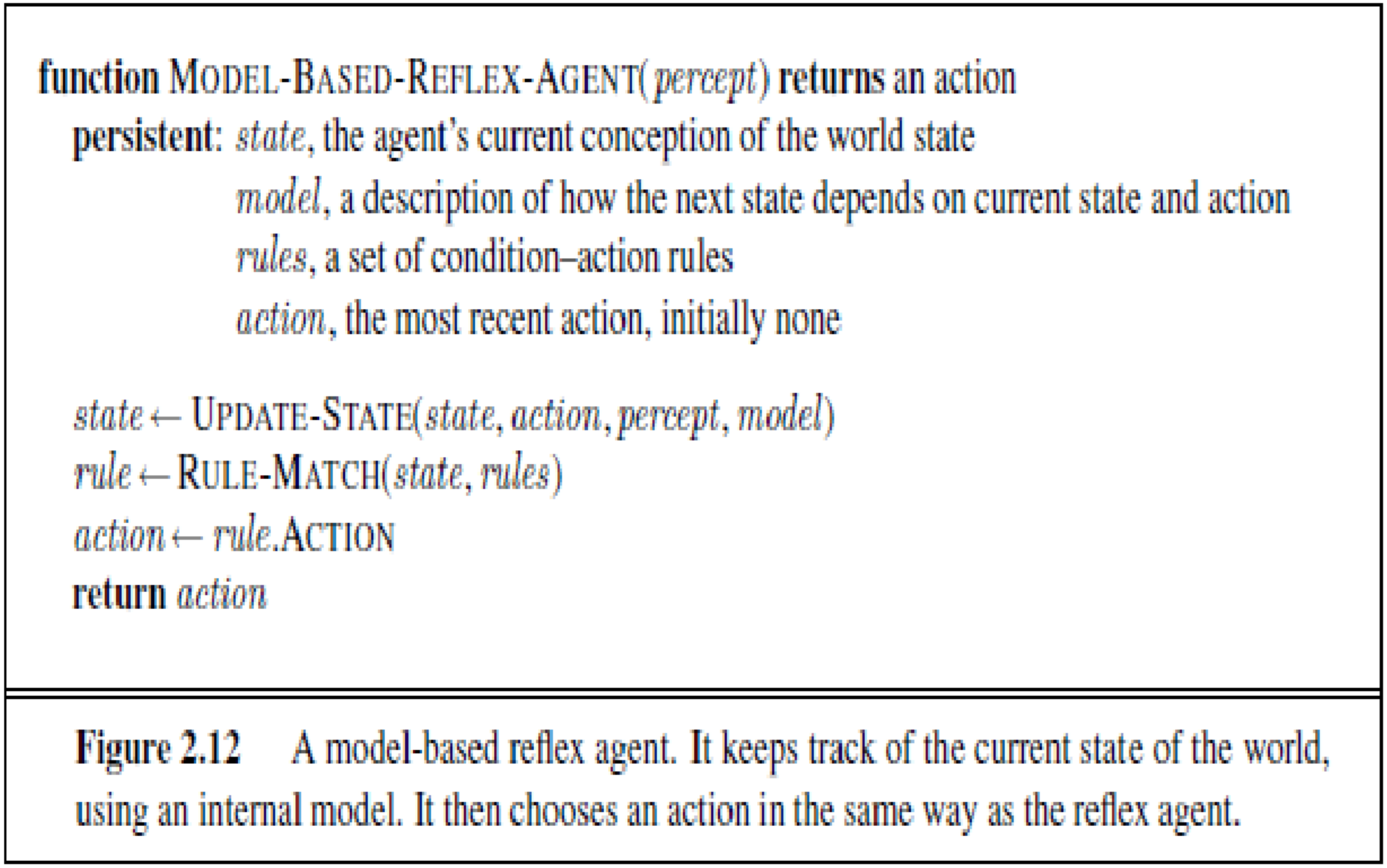
```
function REFLEX-VACUUM-AGENT([location,status]) returns an action

  if status = Dirty then return Suck
  else if location = A then return Right
  else if location = B then return Left
```

Figure 2.8 The agent program for a simple reflex agent in the two-state vacuum environment. This program implements the agent function tabulated in Figure 2.3.

Model-based reflex agents



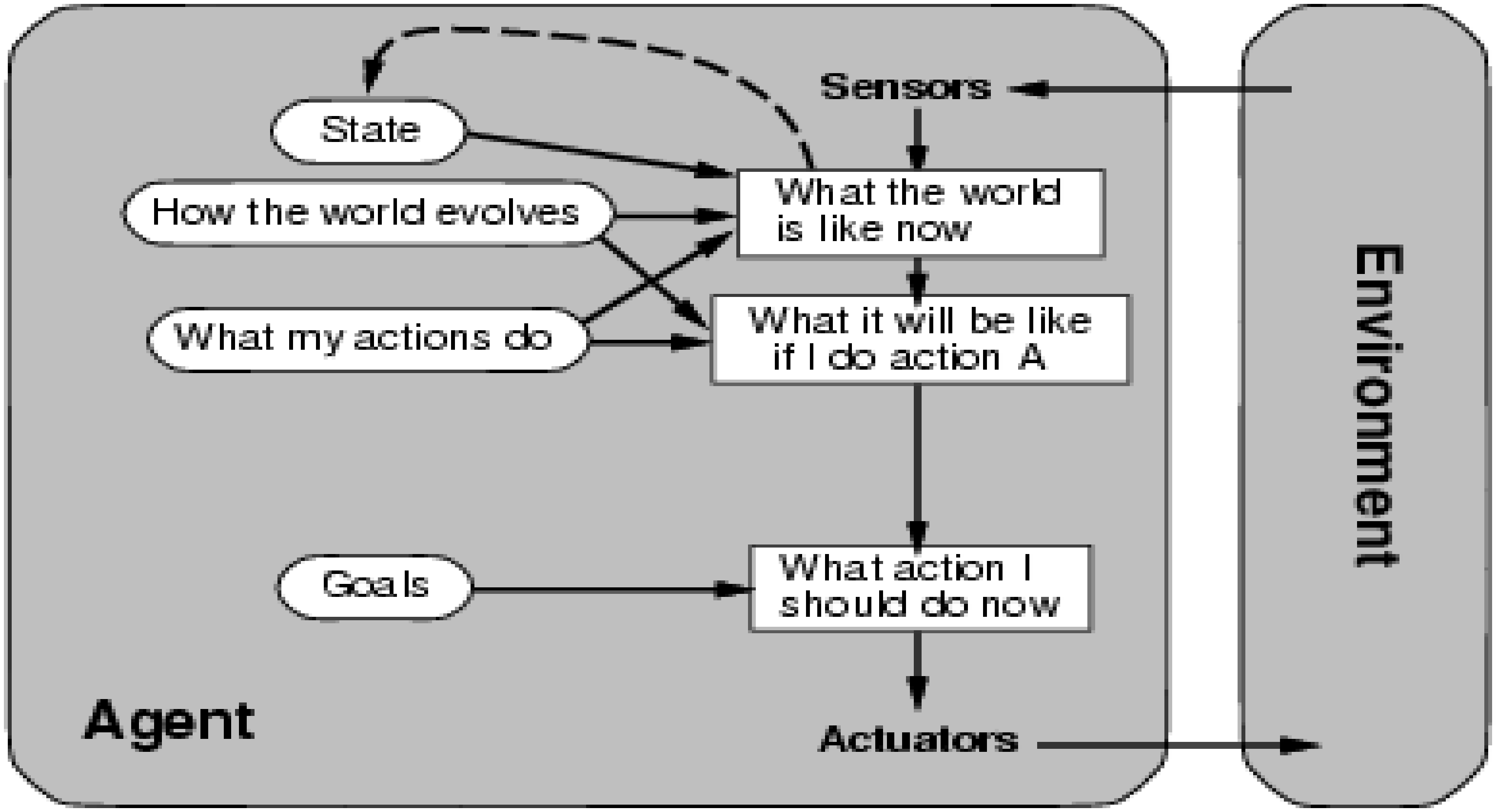
Model-based reflex agents



```
function MODEL-BASED-REFLEX-AGENT(percept) returns an action
  persistent: state, the agent's current conception of the world state
              model, a description of how the next state depends on current state and action
              rules, a set of condition–action rules
              action, the most recent action, initially none

  state ← UPDATE-STATE(state, action, percept, model)
  rule ← RULE-MATCH(state, rules)
  action ← rule.ACTION
  return action
```

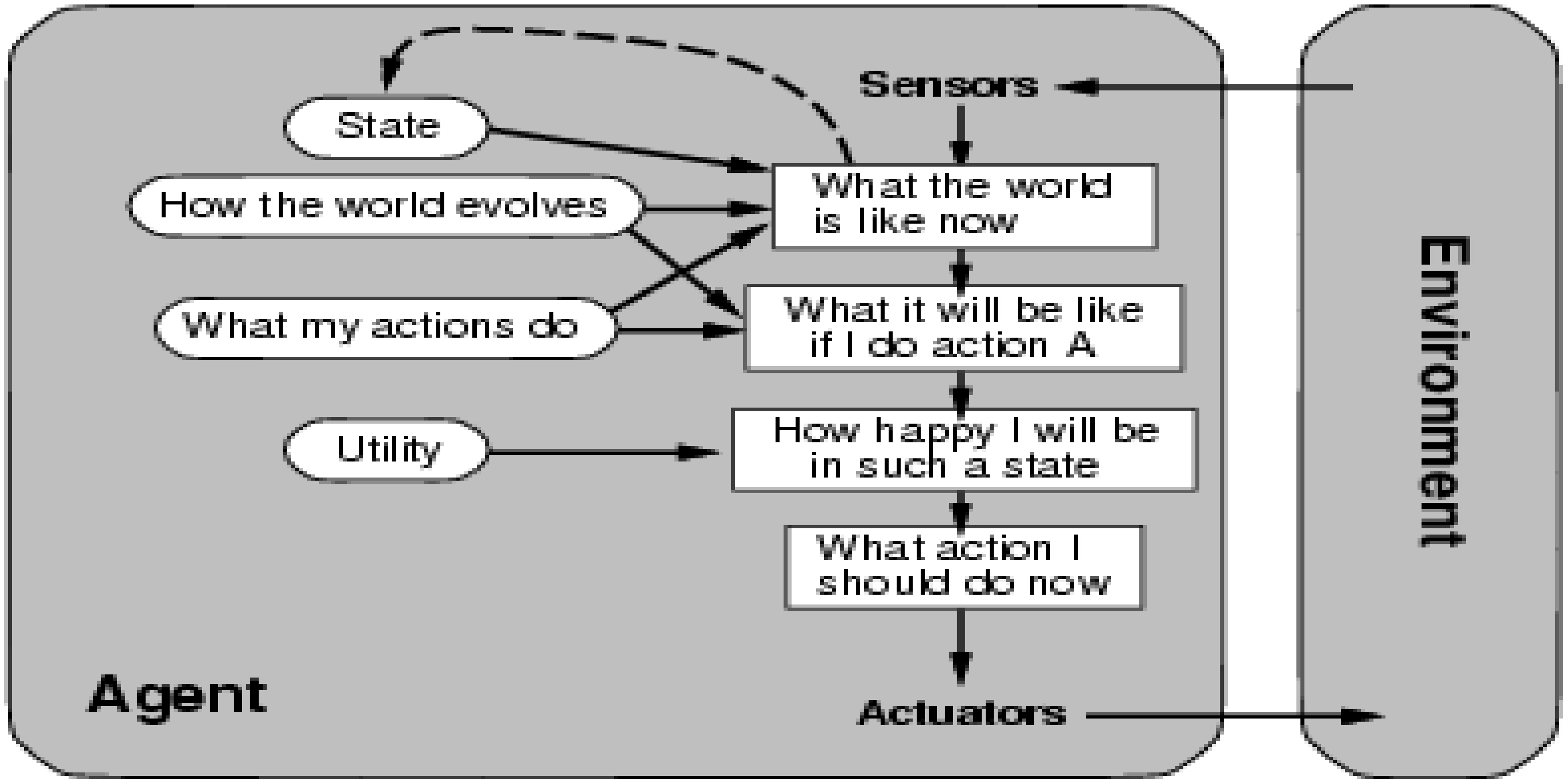
Figure 2.12 A model-based reflex agent. It keeps track of the current state of the world, using an internal model. It then chooses an action in the same way as the reflex agent.

Goal-based agents


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

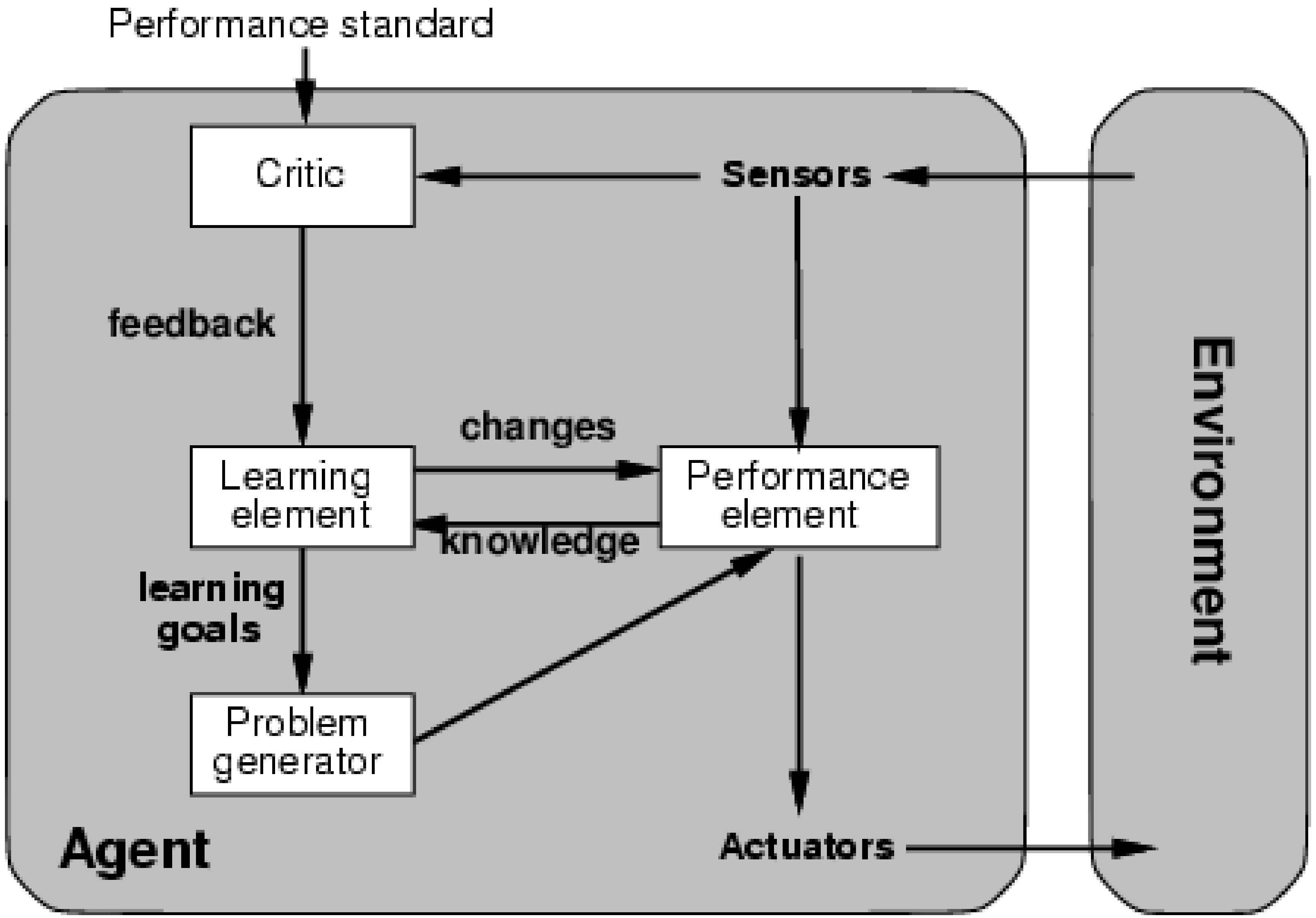
A model-based, goal-based agent. It keeps track of the world state as well as a set of goals it is trying to achieve, and chooses an action that will (eventually) lead to the achievement of its goals. **Search, Planning**.

Utility-based agents


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
Agent
Actuators
Environment

A model-based, utility-based agent. It uses a model of the world, along with a utility function that measures its preferences among states of the world. Then it chooses the action that leads to the best expected utility, where expected utility is computed by averaging over all possible outcome states, weighted by the probability of the outcome.

Learning agents



Why Search?

- To achieve goals or to maximize our utility we need to predict what the result of our actions in the future will be.
- There are many sequences of actions, each with their own utility.
- We want to find, or search for, the best one.

Problem Solving Agent

- Goal Based Agent: Problem-solving agents use **atomic representations** that is, states of the world are considered as wholes, with no internal structure visible to the problem solving algorithms.
- Goal-based agents that use more advanced **factored or structured representations**

  are usually called **planning agents**

Search Algorithms

- **uninformed search algorithms—algorithms that are given no information about the** problem other than its definition.
- Although some of these algorithms can solve any solvable problem, none of them can do so efficiently.
- **Informed search algorithms, on the other hand,** can do quite well given some guidance on where to look for solutions.

Case study : Touring Agent

- Intelligent agents are supposed to maximize their performance measure.
- 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 Romanian, 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 guidebooks.
- Now, suppose the agent has a nonrefundable ticket to fly out of Bucharest the following day.

Touring Agent

- the agent to adopt the **goal of getting to Bucharest**
- Goals help organize behavior by limiting the objectives that the agent is trying to achieve and hence the actions it needs to consider.
- **Goal formulation,** based on the current situation and the agent's performance

measure, is the first step in problem solving.

Touring Agent

- goal : a set of world states—exactly those states
- **Problem formulation** is the process of deciding what actions and states to consider, given a goal.- [unknown , (known)Map]
- *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.*

TA

- *the solution to any problem is a fixed sequence of actions.*
- 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.**

properties of the environment -TA

OK DD

- **Observable:** agent always knows the current state
- **Discrete:**at any given state there are only finitely many actions to choose from.
- **Known:**the agent knows which states are reached by each action.
- **Deterministic:** so each action has exactly one outcome.

summary

- It first formulates a goal and a problem, searches for a sequence of actions that would solve the problem, and then executes the actions one at a time. When this is complete, it formulates another goal and starts over.

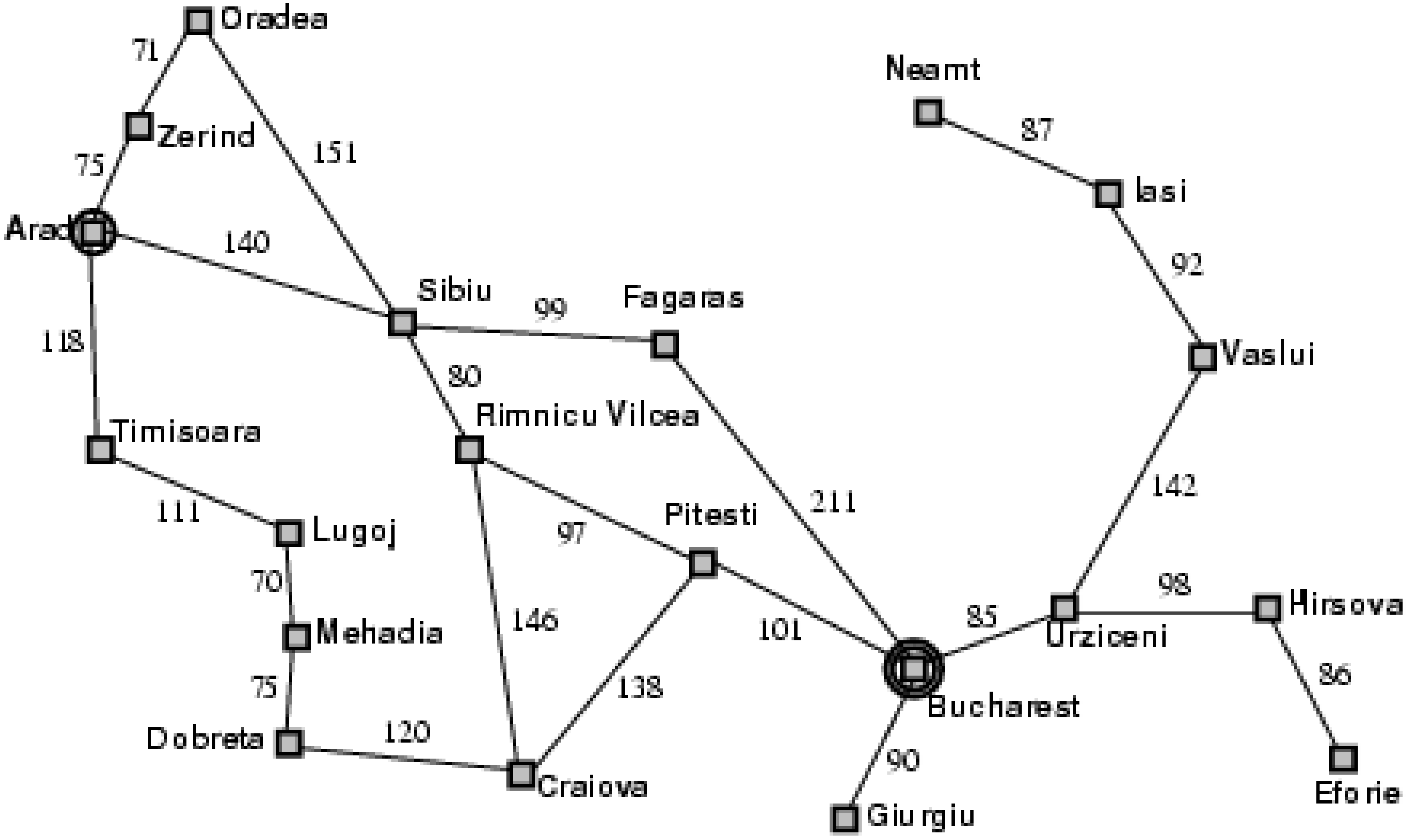
A simple problem-solving agent.

```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
    persistent: seq, an action sequence, initially empty
                state, some description of the current world state
                goal, a goal, initially null
                problem, a problem formulation

    state $\leftarrow$ UPDATE-STATE(state, percept)
    if seq is empty then
        goal $\leftarrow$ FORMULATE-GOAL(state)
        problem $\leftarrow$ FORMULATE-PROBLEM(state, goal)
        seq $\leftarrow$ SEARCH(problem)
        if seq = failure then return a null action
    action $\leftarrow$ FIRST(seq)
    seq $\leftarrow$ REST(seq)
    return action
```

Example: Romania

- On holiday in Romania; currently in Arad.
- Flight leaves tomorrow from Bucharest
- Formulate goal:
  - be in Bucharest
- Formulate problem:
  - states: various cities
  - actions: drive between cities or choose next city
- Find solution:
  - sequence of cities, e.g., Arad, Sibiu, Fagaras, Bucharest

Example: Romania



Problem Formulation

A problem is defined by 5 components:

1.initial state : (particular state s) the agent starts in. e.g., In(Arad)
2.actions / Successor :ACTIONS(s) *S(x)* = set of action–state pairs

the applicable actions are In(Arad), {Go(Sibiu), Go(Timisoara), Go(Zerind)}.

3.Transition model: RESULT(s, a) that returns the state that results from doing action a in state s.

successor **:** to refer to any state reachable from a given state by a single action
RESULT(In(Arad),Go(Zerind)) = In(Zerind)

State Space : 1,2 : 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.

Problem Formulation

A path in the state space is a sequence of states connected by a sequence of actions.

4. goal test :which determines whether a given state is a goal state.
   Ex: 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.

5. path cost :function that assigns a numeric cost to each path.
   – e.g., sum of distances, number of actions executed, etc.
   – c(x ,a, y) is the step cost, assumed to be ≥ 0

A solution is a sequence of actions leading 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

Problem types

- Static / Dynamic

  Previous problem was static: no attention to changes in environment
- Observable / Partially Observable / Unobservable

  Previous problem was observable: it knew initial state.
- Deterministic / Stochastic

  Previous problem was deterministic: no new percepts were necessary, we can predict the future perfectly given our actions
- Discrete / continuous

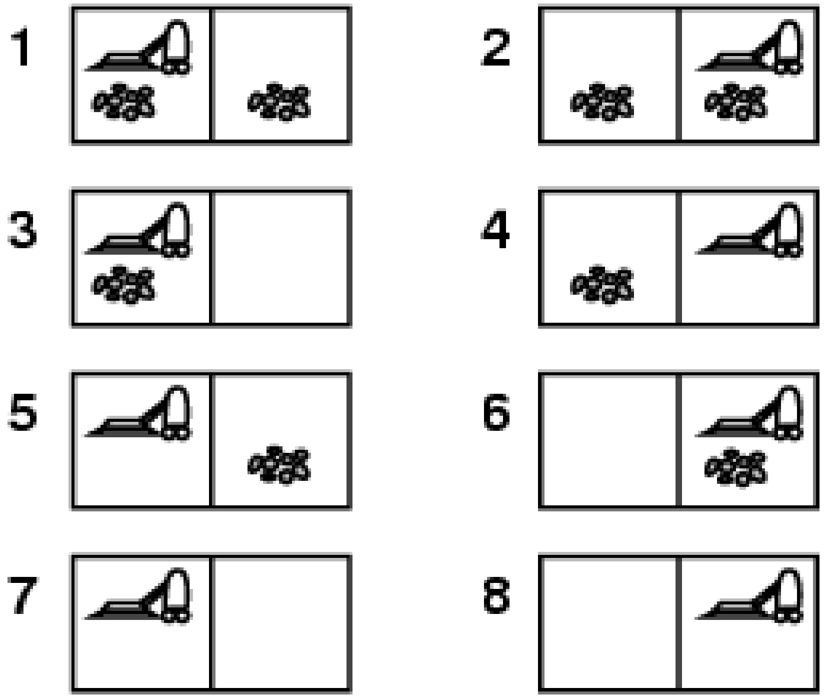
  Previous problem was discrete: we can enumerate all possibilities

Example: vacuum world

- states? discrete: dirt and robot
- Environment with n locations has $n2^n$ states.
- initial state? Any
- actions? *Left*, *Right*, *Suck*
- 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? no dirt at all locations
- path cost? 1 per action

Example: vacuum world

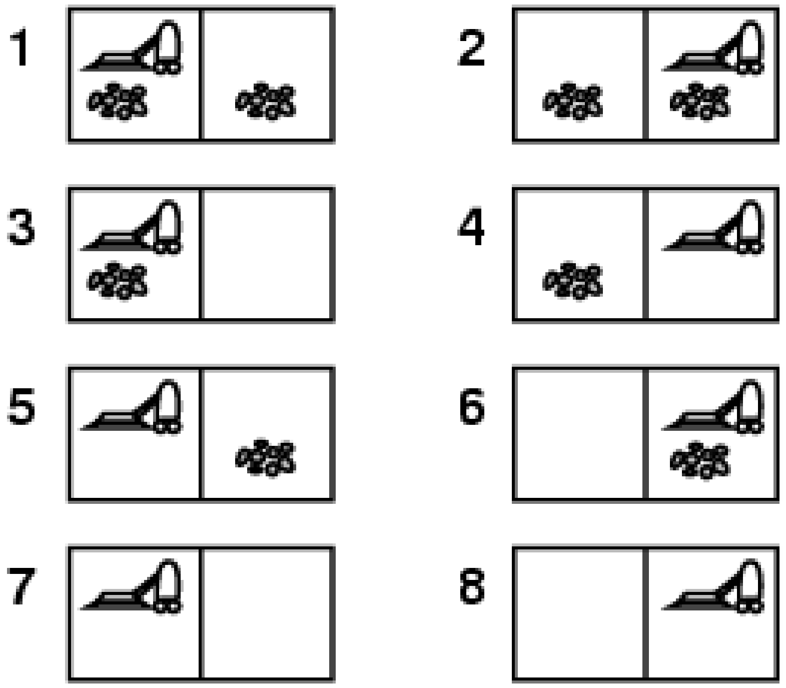
- Observable, start in #5. Solution?
- $n2^n$
- 2×2^2



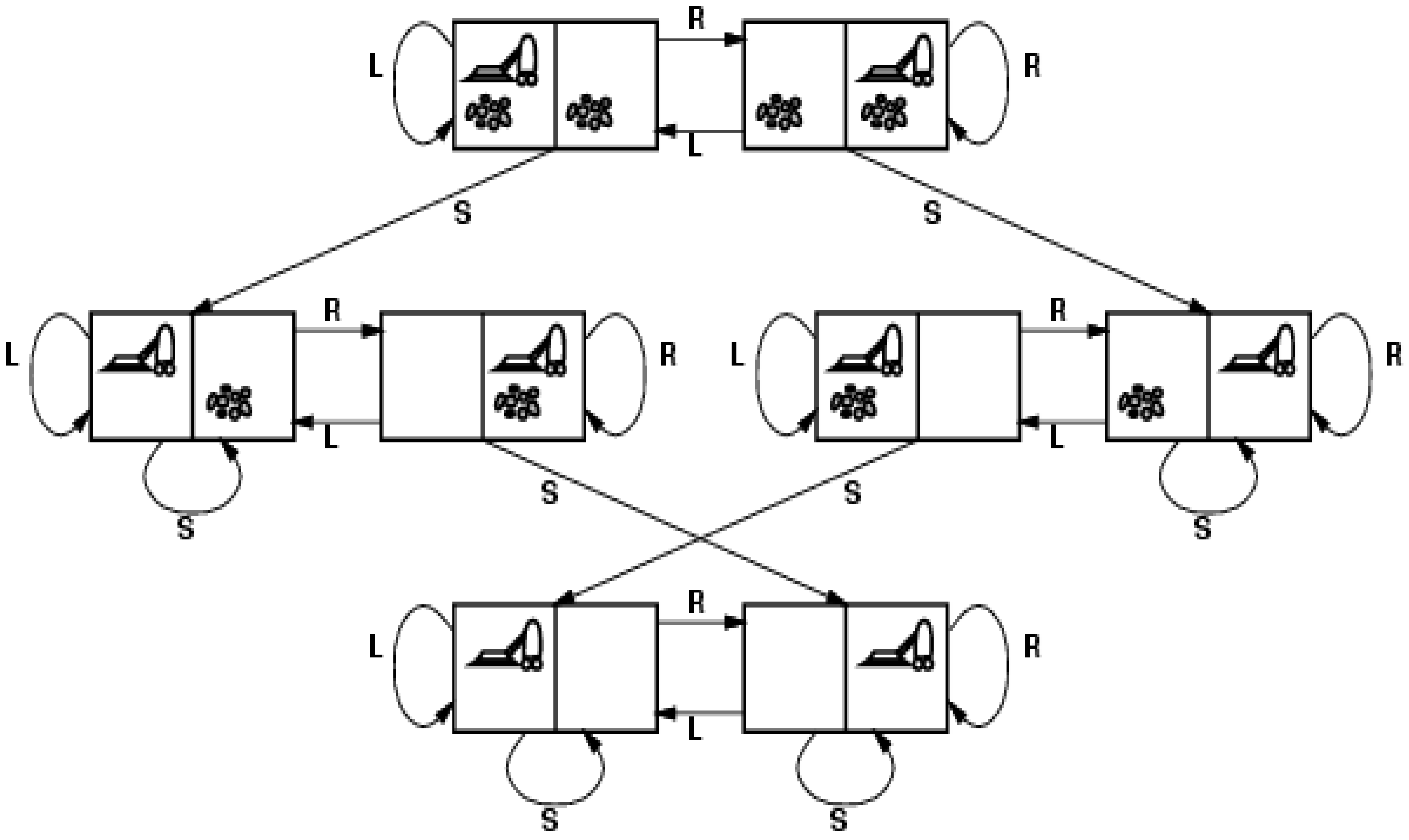
Example: vacuum world

- Observable, start in #5.
  Solution? *[Right, Suck]*

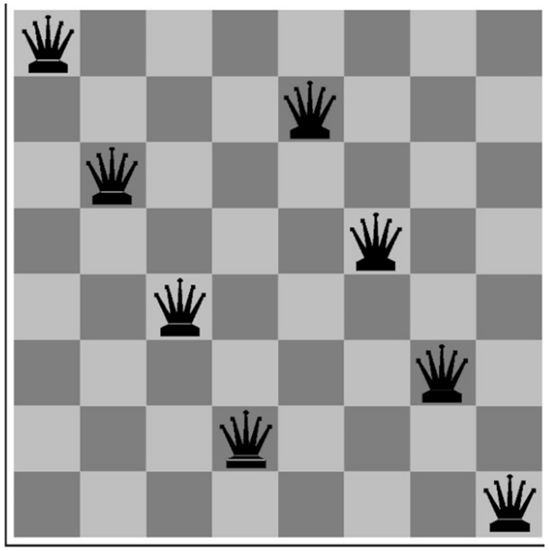
- Unobservable, start in {*1,2,3,4,5,6,7,8*} e.g.,
  Solution?



Vacuum world state space graph



Example: 8-Que



- states? -any arrangement of n<=8 queens
  -*or* arrangements of n<=8 queens in leftmost n columns, 1 per column, such that no queen attacks any other.
- initial state? **incremental formulation :** no queens on the board
  complete-state formulation starts with all 8 queens on the board

What is nPr and nCr in math?

- In Maths, nPr and nCr are the probability functions that represent **permutations and combinations.**
- The method of representation of a group of objects and arranging them in various ways, is described by permutations and combinations
- **Permutation:** A permutation is the arrangements of r things from a set of n things without replacement. Order matters in the permutation.The formula to find nPr is given by: nPr = n!/(n-r)!
- **Combination:** A combination is the choice of r things from a set of n things without replacement. The order does not matter in combination. The formula to find nCr is: nCr = n!/[r! (n-r)!] = npr/r!

Where n is the total number of objects and r is the number of selected objects.

Here n! is the product of all positive integers less than and equal to n.

- For example, if n = 5, then n! = 5 x 4 x 3 x 2 x 1

8-Queens

- actions? -add queen to any empty square

  -*or* add queen to leftmost empty square such that it is not attacked by other queens. /move the queen around

Transition model? Returns the board with a queen added to the specified square.

- goal test? 8 queens on the board, none attacked.
- path cost? 1 per move

possible sequences to investigate:

1.8×10^{14} possible sequences to investigate.

npr -> **n! = 1.2688693218588E+89 n ----- 64, r -- 8**

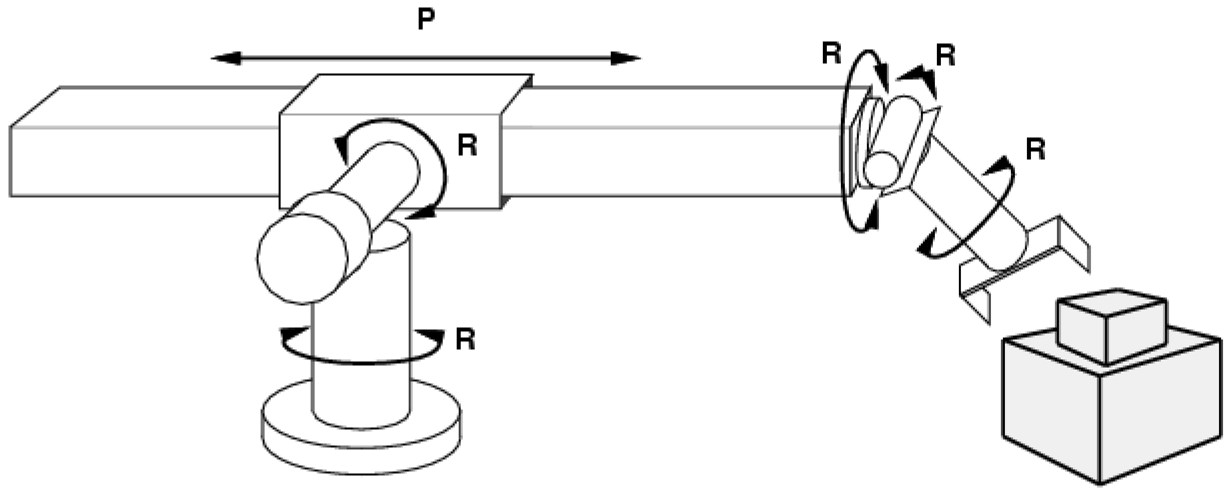
(n-r)! = 7.1099858780486E+74

${}^{n}P_{r}$ = n!/(n-r)!

${}^{n}P_{r}$ = 1.2688693218588E+89/7.1099858780486E+74

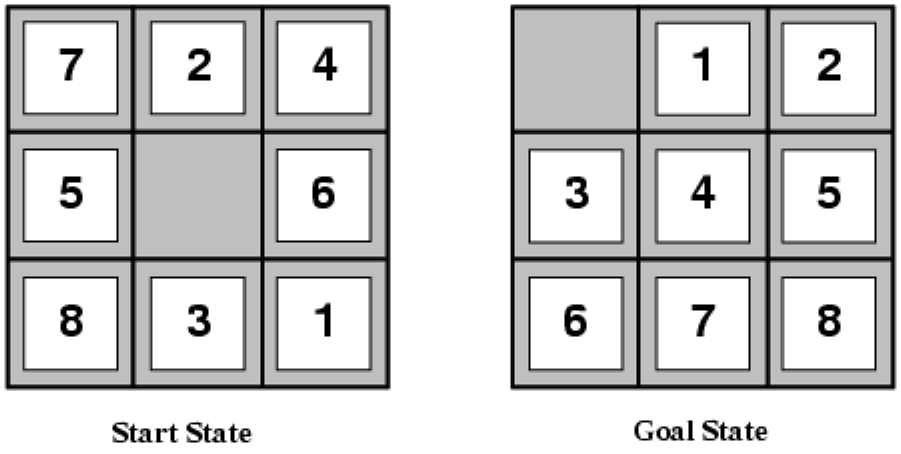
${}^{n}P_{r}$ = 1.7846298763776E+14

Example: robotic assembly



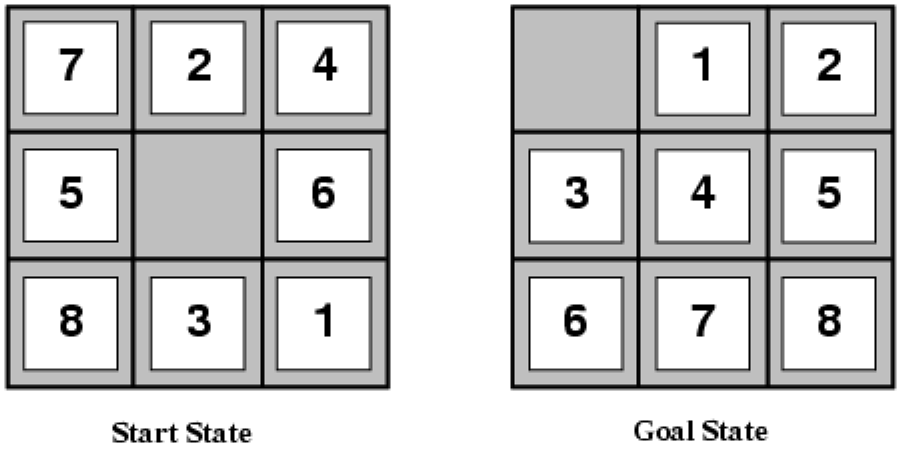
- states?: real-valued coordinates of robot joint angles parts of the object to be assembled
- initial state?: rest configuration
- actions?: continuous motions of robot joints
- goal test?: complete assembly
- path cost?: time to execute

Example: The 8-puzzle



- states?
- initial state?
- actions?
- goal test?
- path cost?

Example: The 8-puzzle



- states? locations of tiles, blank in one.
- initial state? given
- actions? move blank left, right, up, down
- Transitional model: ? Given a state and action, this returns the resulting state
- goal test? goal state (given)
- path cost? 1 per move Abstractions?

The 8-puzzle

- Note that any given goal can be reached from exactly half of the possible initial states

[Note: optimal solution of *n*-Puzzle family is NP-hard]

- The 8-puzzle has 9!/2=181, 440 reachable states and is easily solved.

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.

Tree search algorithms

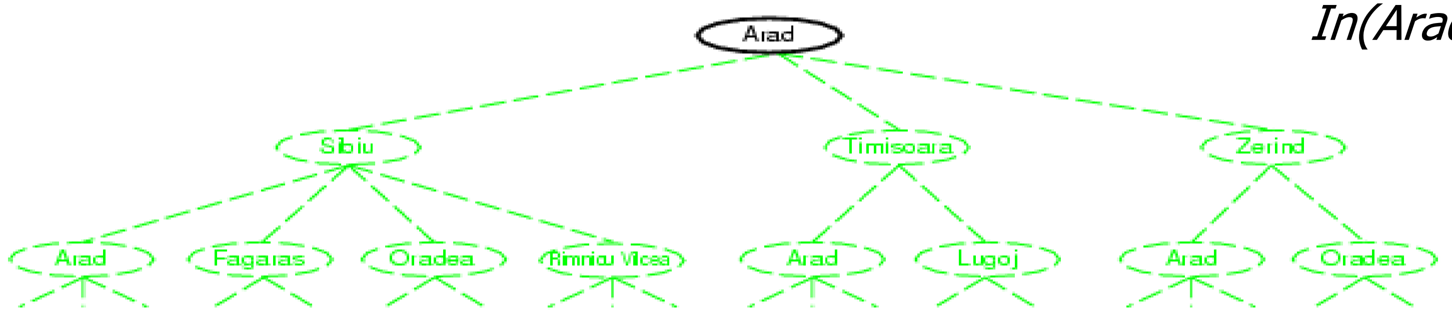
- Basic idea:
  - Exploration of state space by generating successors of already-explored states (a.k.a.~expanding states).

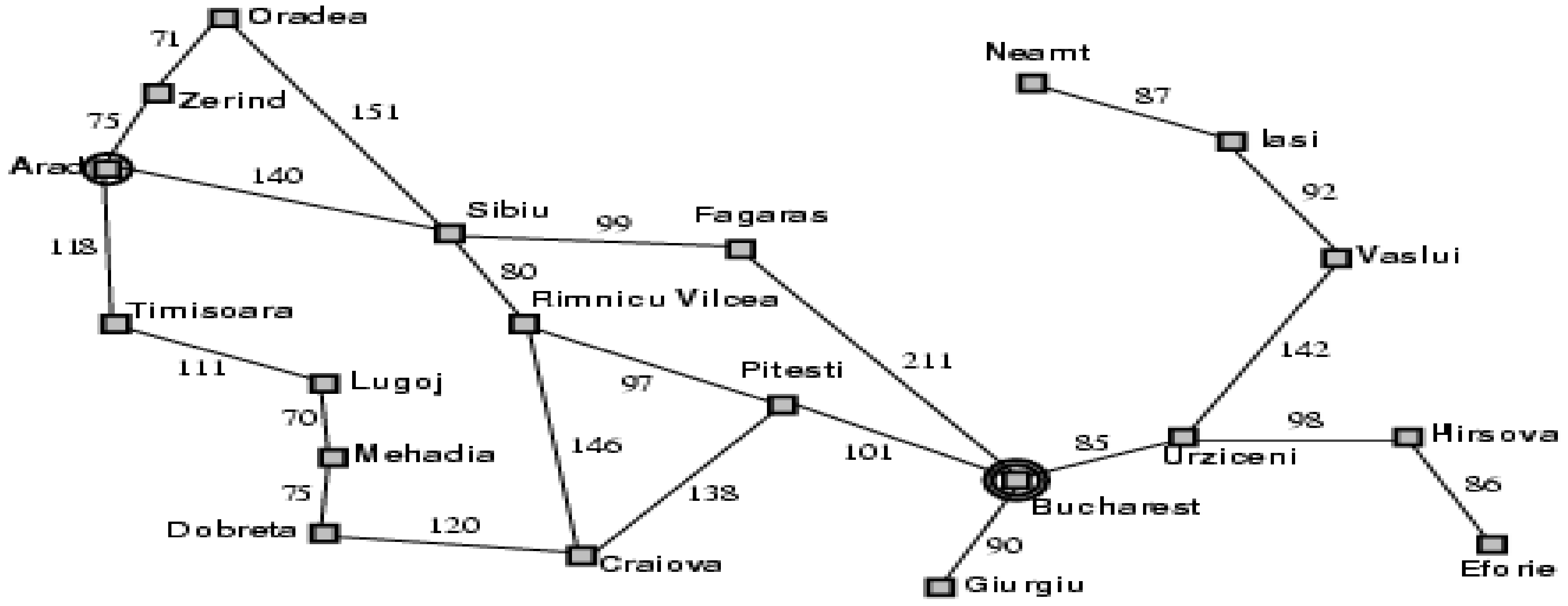
  - Every states is evaluated: *is it a goal state*?
- Nodes that have been expanded are shaded
- Nodes that have been generated but not yet expanded are

  outlined in bold
- nodes that have not yet been generated are shown in faint dashed lines.

Tree search example

In(Arad)

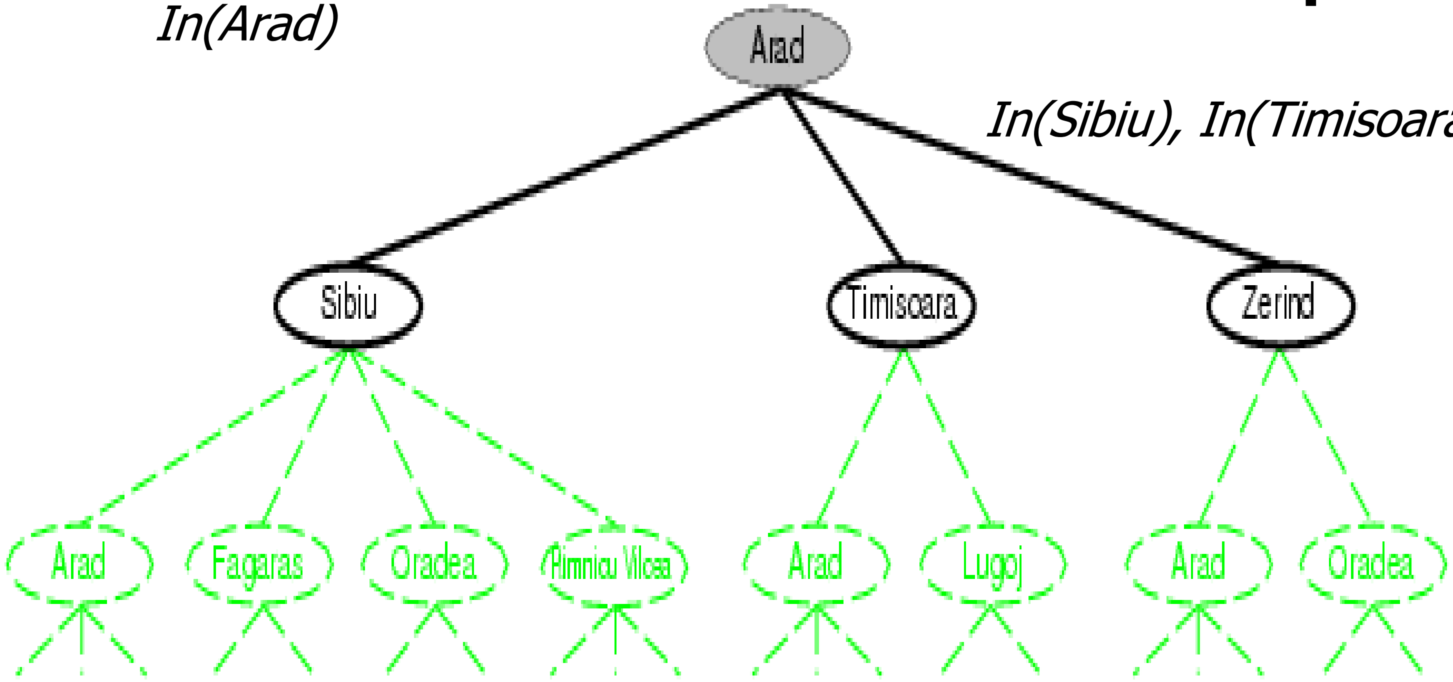


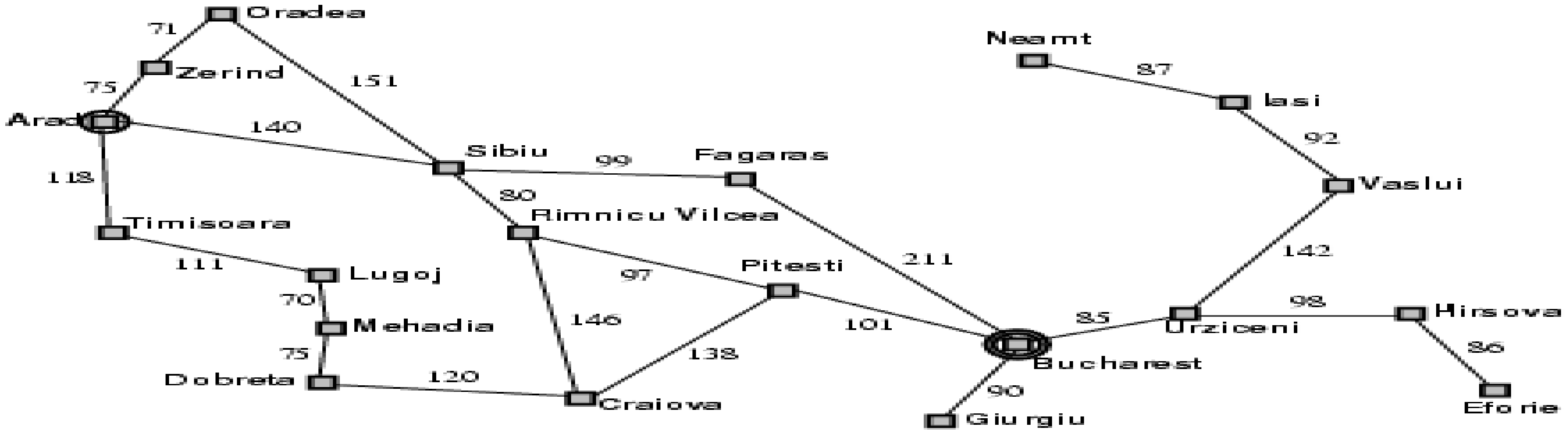


Tree search example

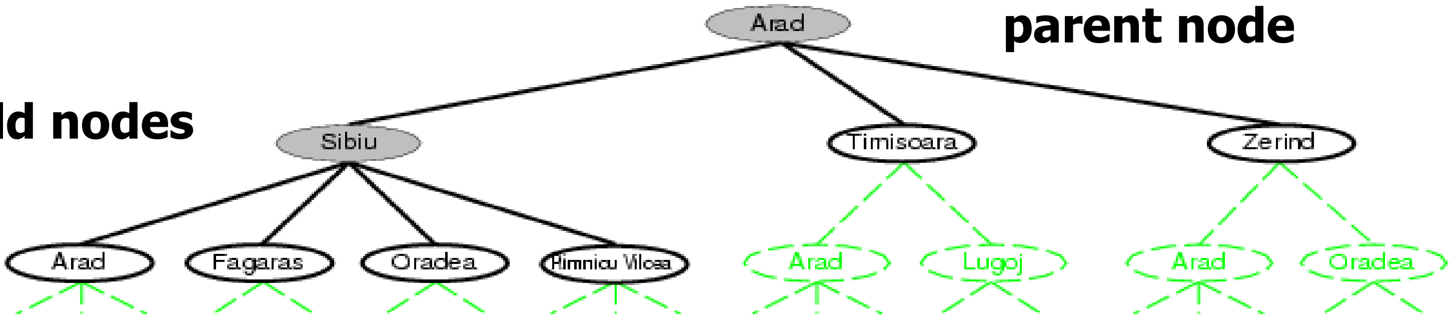
In(Arad)

In(Sibiu), In(Timisoara), and In(Zerind).





Tree search example



parent node

Child nodes

goal state ?

In(Arad), In(Fagaras), In(Oradea), and In(RimnicuVilcea).

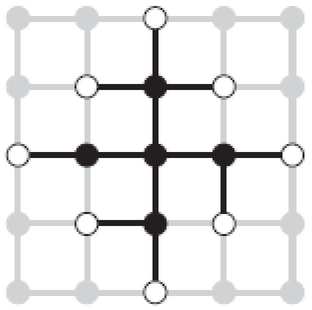
leaf node : a node with no children in the tree

The set of all leaf nodes available for expansion at any given point is called the **frontier.** Also called as **open list.**

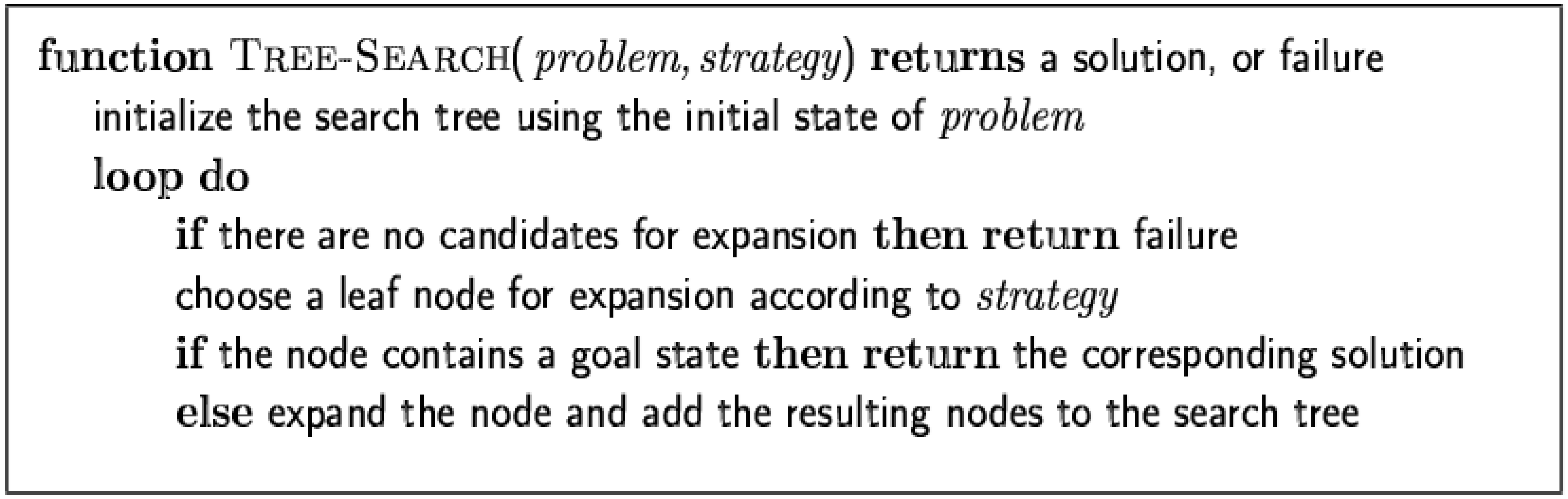
how they choose which state to expand next—**search strategy**

In(Arad) is a **repeated state** in the search tree**,** generated in this case by a **loopy path. - redundant paths.**

- redundant paths are unavoidable. This includes all problems where the actions are reversible, such as route-finding problems and sliding-block puzzles. *algorithms that forget their history are doomed to repeat it.*
- The way to avoid exploring redundant paths is to remember where one has been. To do this, we 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 discarded instead of being added to the frontier.



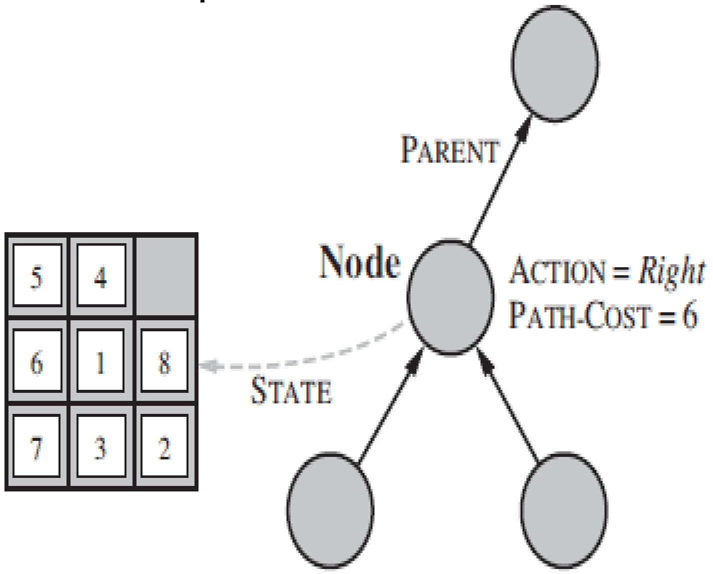
Tree Search Algorithm



```
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
```

Infrastructure for search algorithms

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



Arrows point from child to parent.

Implementation

- **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)

- The appropriate data structure for this is a **queue.**
- **The** operations on a queue are as follows:
- • EMPTY?(queue) returns true only if there are no more elements in the queue.
- • POP(queue) removes the first element of the queue and returns it.
- • INSERT(element, queue) inserts an element and returns the resulting queue.

- **FIFO queue, LIFO queue, Priority queue**

Measuring problem-solving algorithm performance

- A search strategy is defined by picking the order of node expansion
- Strategies are evaluated along the following dimensions:
  - completeness: does it always find a solution if one exists?
  - time complexity: number of nodes generated
  - space complexity: maximum number of nodes in memory
  - optimality: does it always find a least-cost solution?
- Time and space complexity are measured in terms of
  - *b:* maximum branching factor of the search tree
  - *d:* depth of the least-cost solution
  - *m*: maximum depth of the state space (may be ∞)

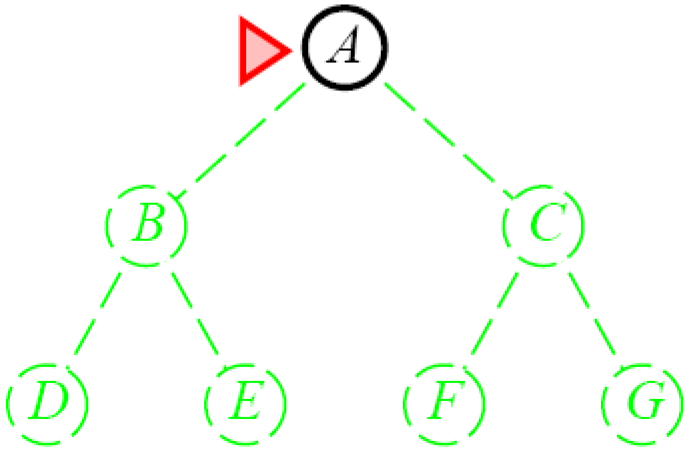
Search Strategies

- The UNINFORMED SEARCH / BLIND SEARCH 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 are called **informed search or heuristic search strategies;**

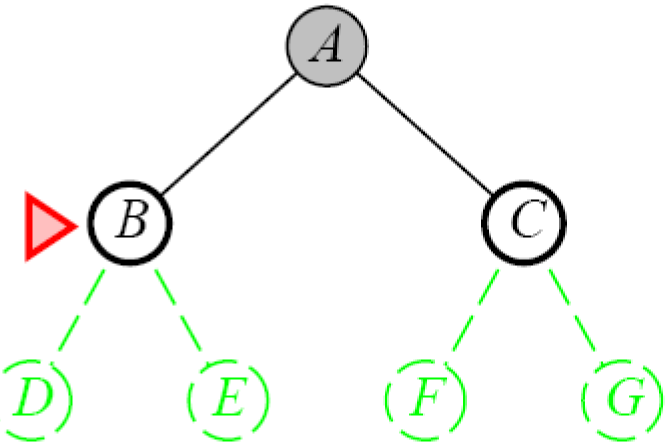
UNINFORMED SEARCH Strategies

- Breadth-first search
- Uniform-cost search
- Depth-first search
- Depth-limited search
- Iterative deepening depth-first search
- Bidirectional search

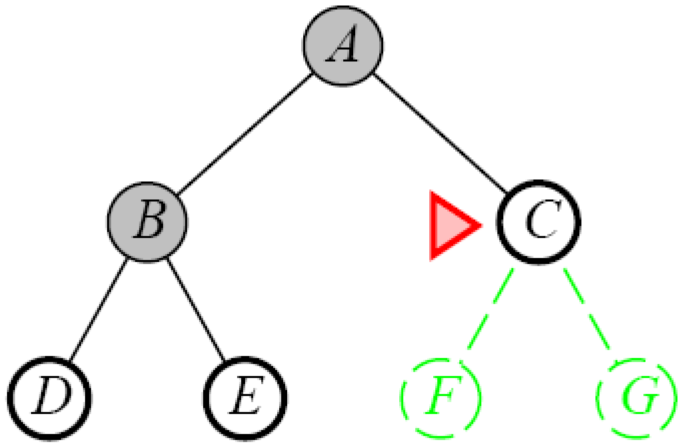
Breadth-First Search



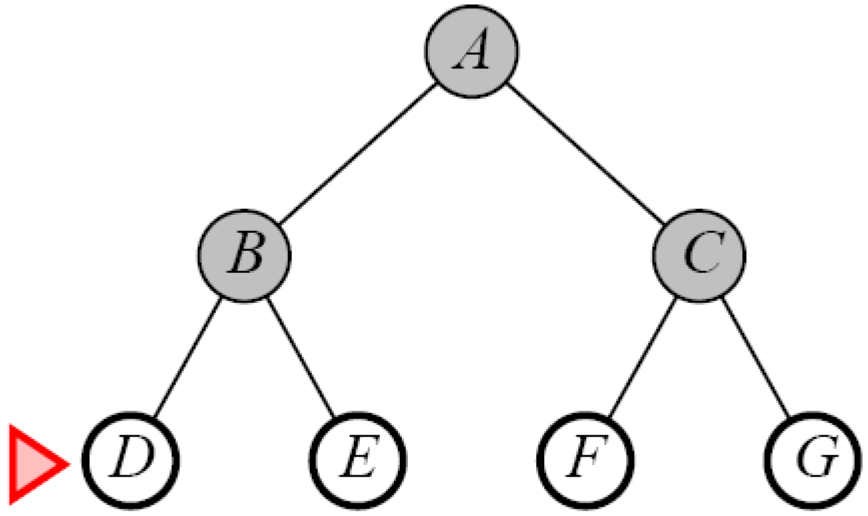
Breadth-First Search



Breadth-First Search



Breadth-First Search



Breadth-First

- Enqueue nodes on nodes in **FIFO** (first-in, first-out) order.
- **Complete**
- **Optimal** (i.e., admissible) if all operators have the same cost. Otherwise, not optimal but finds solution with shortest path length.
- **Exponential time and space complexity**, $O(b^d)$, where d is the depth of the solution and b is the branching factor (i.e., number of children) at each node
- Will take a **long time to find solutions** with a large number of steps because must look at all shorter length possibilities first
  - A complete search tree of depth d where each non-leaf node has b children, has a total of $1 + b + b^2 + ... + b^d = (b^{(d+1)} - 1)/(b-1)$ nodes
  - For a complete search tree of depth 12, where every node at depths 0, ..., 11 has 10 children and every node at depth 12 has 0 children, there are $1 + 10 + 100 + 1000 + ... + 10^{12} = (10^{13} - 1)/9 = O(10^{12})$ nodes in the complete search tree. If BFS expands 1000 nodes/sec and each node uses 100 bytes of storage, then BFS will take 35 years to run in the worst case, and it will use 111 terabytes of memory!

Uniform-Cost Search

- Same idea as the algorithm for breadth-first search…but…
  - Expand the ***least-cost*** unexpanded node
  - FIFO queue is ordered by cost
  - Equivalent to regular breadth-first search if all step costs are equal

Uniform-Cost Search

- Complete
  - Yes if the cost is greater than some threshold
  - step cost >= ε
- Time
  - Complexity cannot be determined easily by d or d
  - Let C* be the cost of the optimal solution
  - $O(b^{ceil(C^*/ \varepsilon)})$
- Space
  - $O(b^{ceil(C^*/ \varepsilon)})$
- Optimal
  - Yes, Nodes are expanded in increasing order

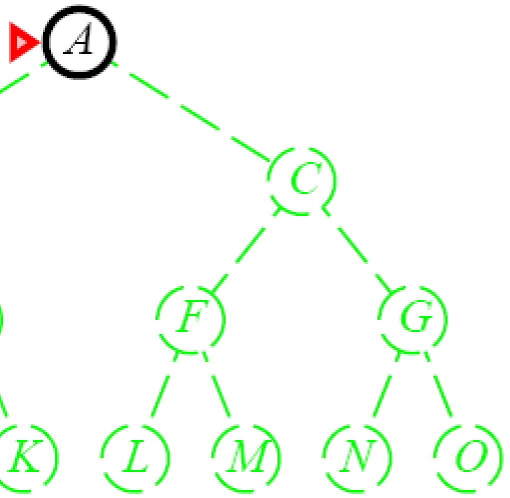
Uniform-Cost (UCS)

- Enqueue nodes by path cost. That is, let g(n) = cost of the path from the start node to the current node n. Sort nodes by increasing value of g.
- Called “Dijkstra’s Algorithm” in the algorithms literature and similar to “Branch and Bound Algorithm” in operations research literature
- Complete (*)
- Optimal/Admissible (*)
- Admissibility depends on the goal test being applied when a node is removed from the nodes list, not when its parent node is expanded and the node is first generated
- Exponential time and space complexity, O(bd)

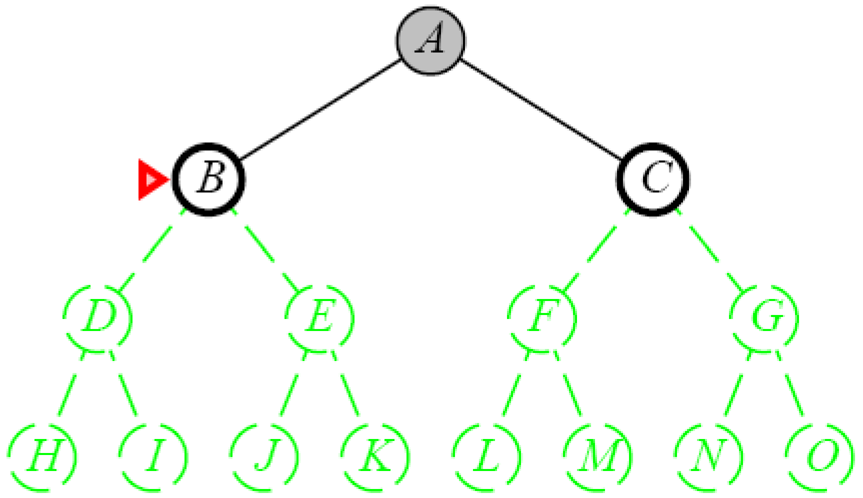
Depth-First Search

- Recall from Data Structures the basic algorithm for a depth-first search on a graph or tree
- Expand the ***deepest*** unexpanded node
- Unexplored successors are placed on a stack until fully explored

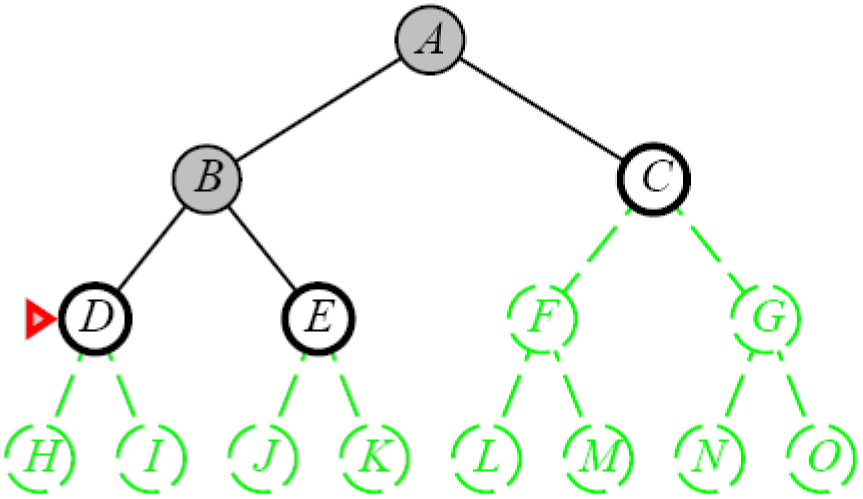
Depth-First Search



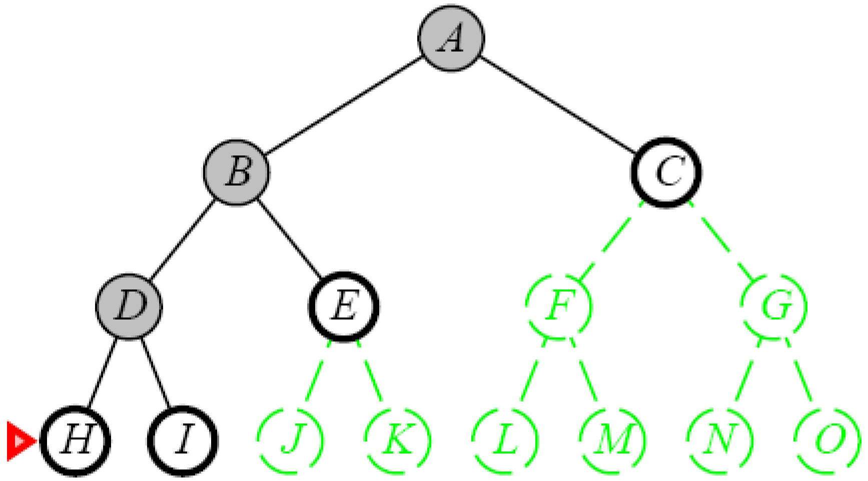
Depth-First Search



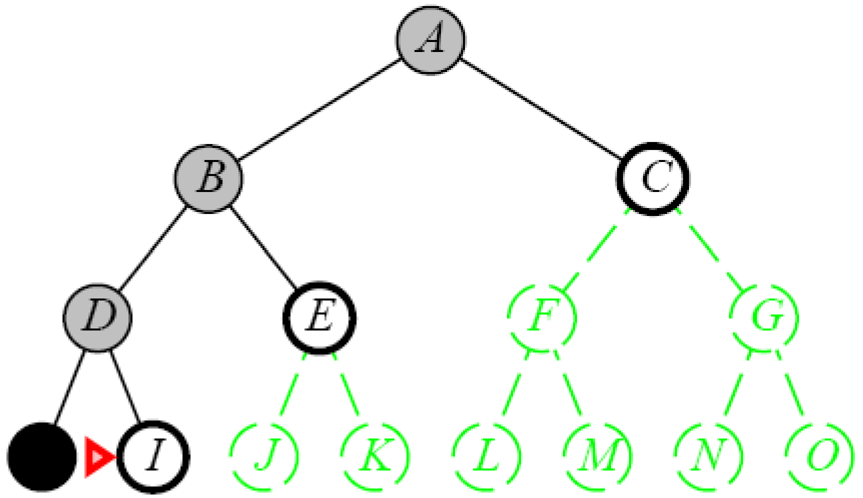
Depth-First Search



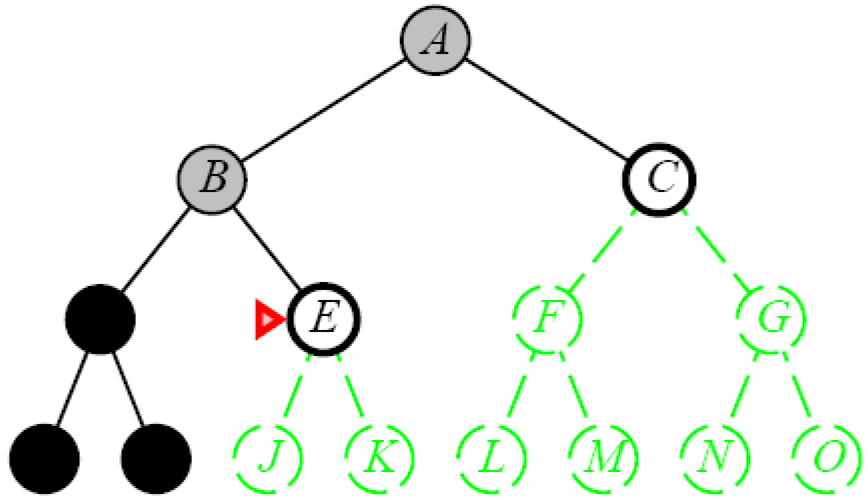
Depth-First Search



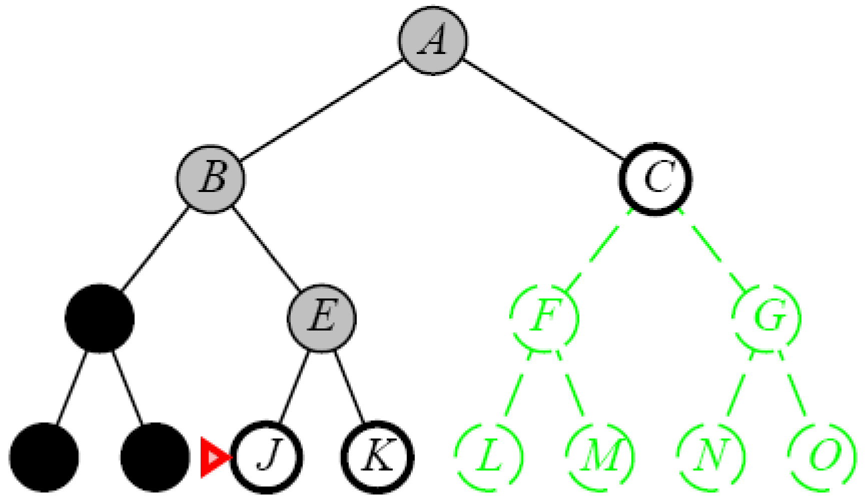
Depth-First Search



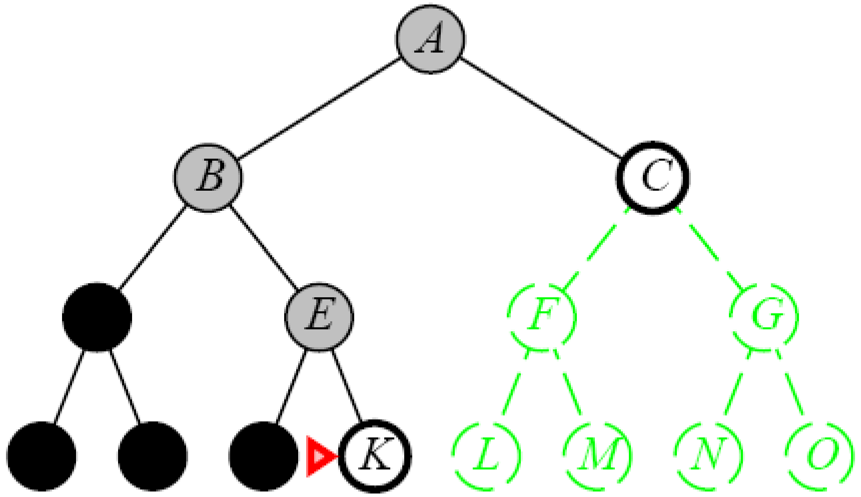
Depth-First Search



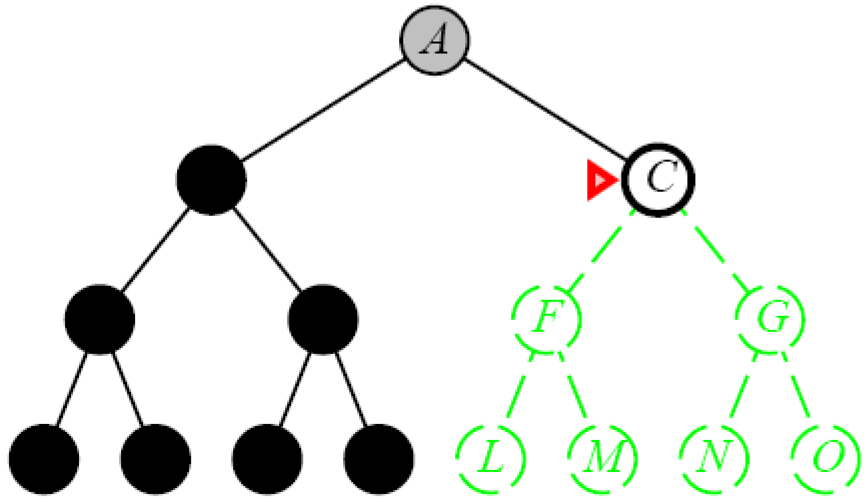
Depth-First Search



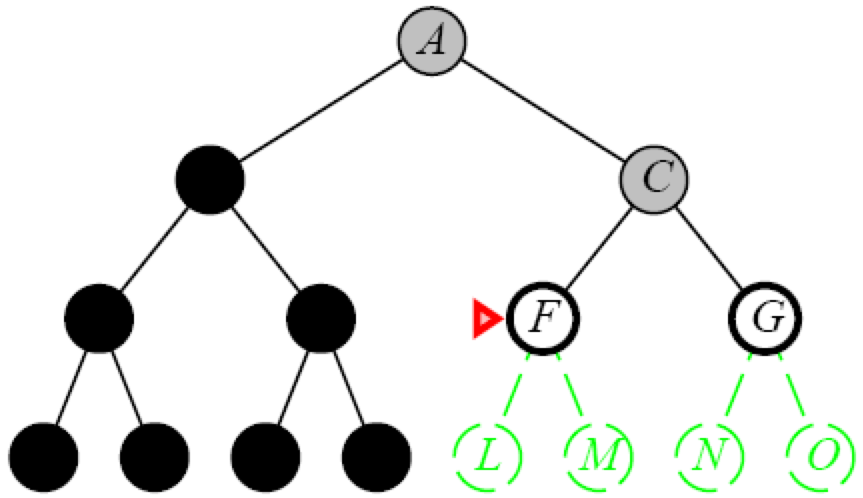
Depth-First Search



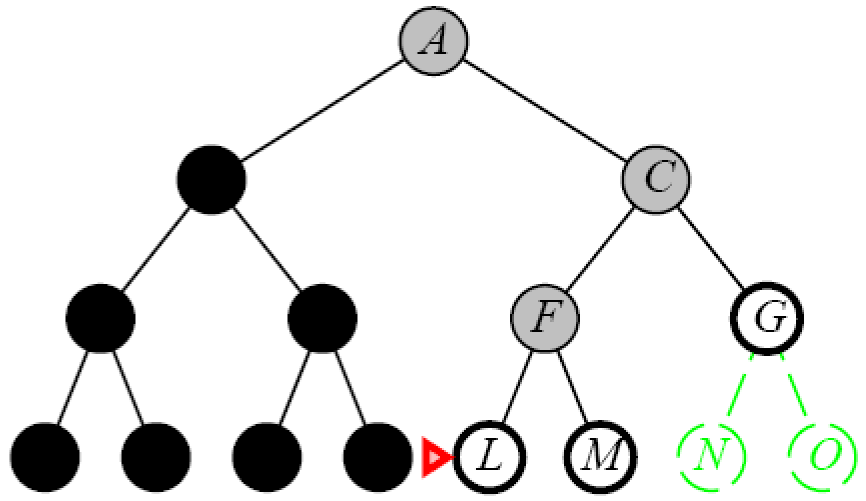
Depth-First Search



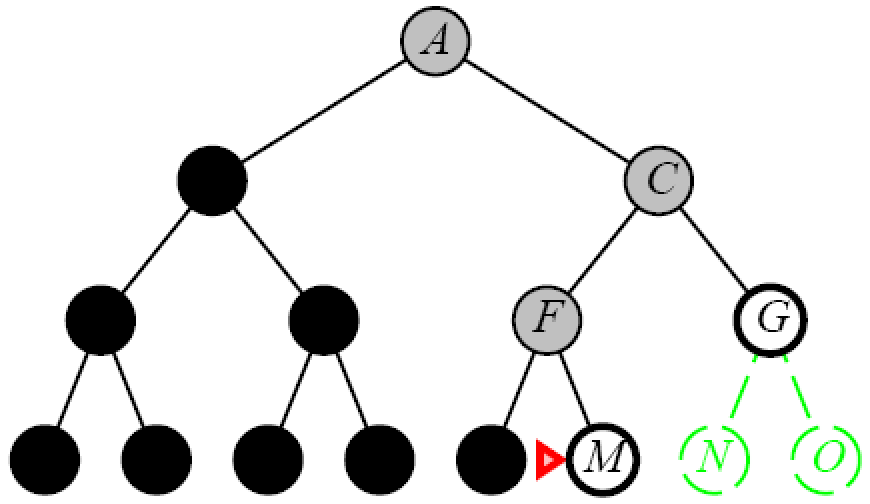
Depth-First Search



Depth-First Search



Depth-First Search



Depth-First Search

- Complete
  - No: fails in infinite-depth spaces, spaces with loops
    - Modify to avoid repeated spaces along path
  - Yes: in finite spaces
- Time
  - $O(b^m)$
  - Not great if m is much larger than d
  - But if the solutions are dense, this may be faster than breadth-first search
- Space
  - O(bm)…linear space
- Optimal
  - No

Depth-First (DFS)

- Enqueue nodes on nodes in LIFO (last-in, first-out) order. That is, nodes used as a stack data structure to order nodes.
- May not terminate without a “depth bound,” i.e., cutting off search below a fixed depth D (“depth-limited search”)
- Not complete (with or without cycle detection, and with or without a cutoff depth)
- Exponential time, O(bd), but only linear space, O(bd)
- Can find long solutions quickly if lucky (and short solutions slowly if unlucky!)
- When search hits a dead-end, can only back up one level at a time even if the “problem” occurs because of a bad operator choice near the top of the tree. Hence, only does “chronological backtracking”

Depth-Limited Search

- A variation of depth-first search that uses a depth limit
  - Alleviates the problem of unbounded trees
  - Search to a predetermined depth **l** (“ell”)
  - Nodes at depth l have no successors

- Same as depth-first search if l = ∞
- Can terminate for failure and cutoff

Depth-Limited Search

Recursive implementation:

```
function DEPTH-LIMITED-SEARCH(problem, limit) returns soln/fail/cutoff
    RECURSIVE-DLS(MAKE-NODE(INITIAL-STATE[problem]), problem, limit)

function RECURSIVE-DLS(node, problem, limit) returns soln/fail/cutoff
    cutoff-occurred? ← false
    if GOAL-TEST[problem](STATE[node]) then return node
    else if DEPTH[node] = limit then return cutoff
    else for each successor in EXPAND(node, problem) do
        result ← RECURSIVE-DLS(successor, problem, limit)
        if result = cutoff then cutoff-occurred? ← true
        else if result ≠ failure then return result
    if cutoff-occurred? then return cutoff else return failure
```

Depth-Limited Search

- Complete
  - Yes if l < d
- Time
  - $O(b^l)$
- Space
  - $O(bl)$
- Optimal
  - No if l > d

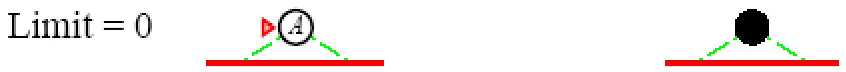
Iterative Deepening Search

- Iterative deepening depth-first search
  - Uses depth-first search
  - Finds the best depth limit
    - Gradually increases the depth limit; 0, 1, 2, … until a goal is found

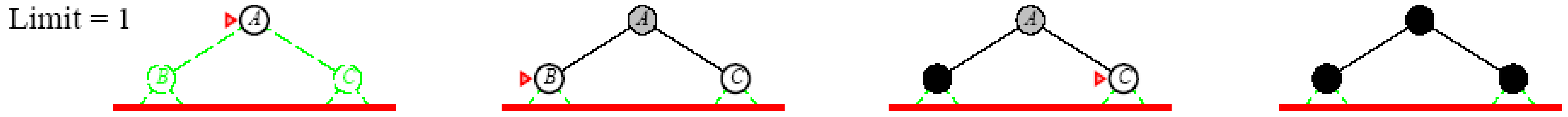
Iterative Deepening Search

```
function ITERATIVE-DEEPENING-SEARCH(problem) returns a solution
    inputs: problem, a problem

    for depth ← 0 to ∞ do
        result ← DEPTH-LIMITED-SEARCH(problem, depth)
        if result ≠ cutoff then return result
    end
```

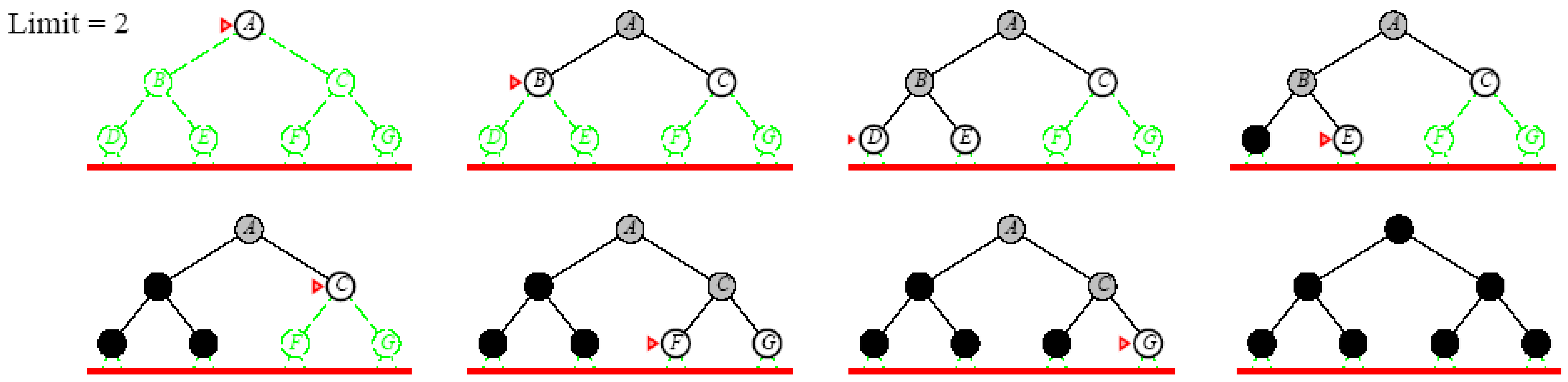
Iterative Deepening Search



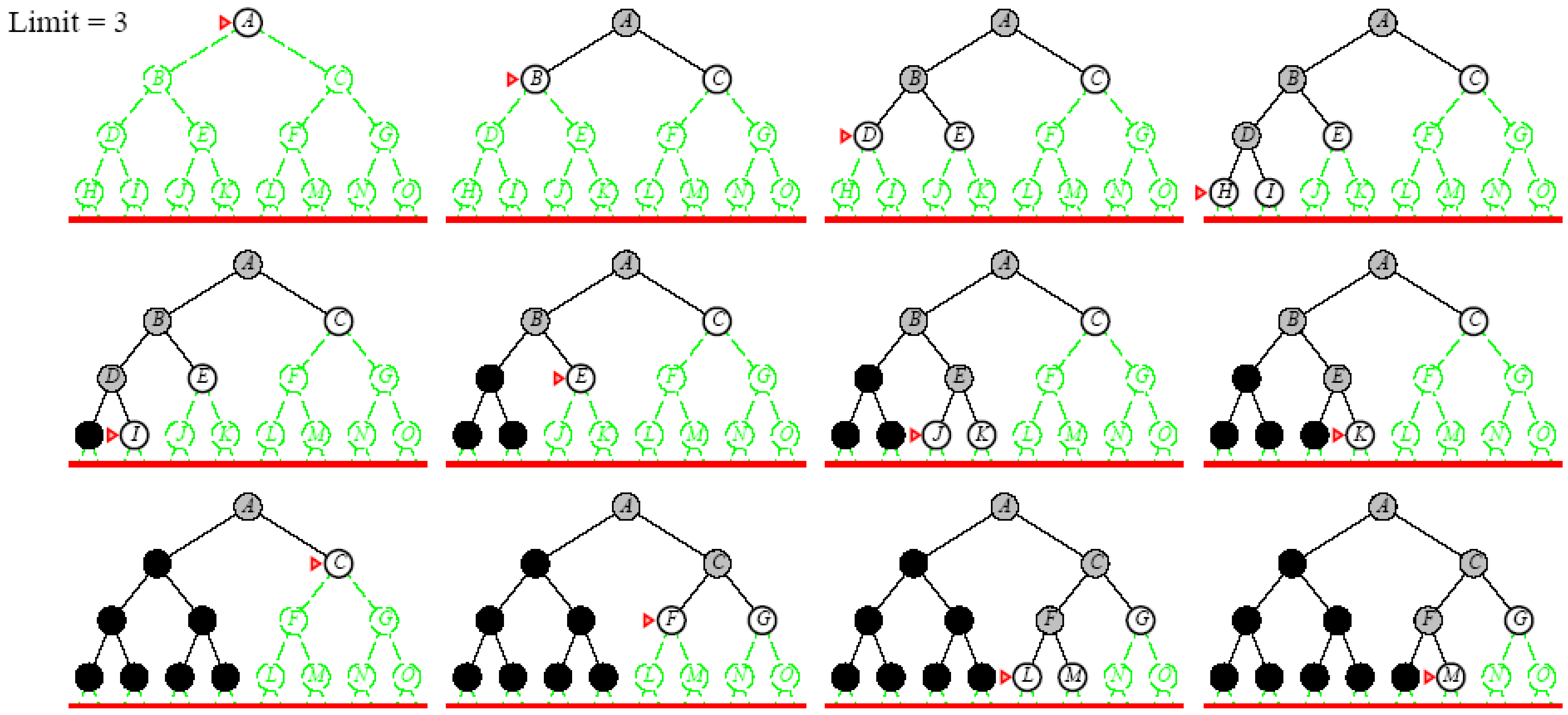
Iterative Deepening Search



Iterative Deepening Search



Iterative Deepening Search



Iterative Deepening Search

- Complete
  - Yes
- Time
  - $O(b^d)$
- Space
  - $O(bd)$
- Optimal
  - Yes if step cost = 1
  - Can be modified to explore uniform cost tree

Lessons From Iterative Deepening Search

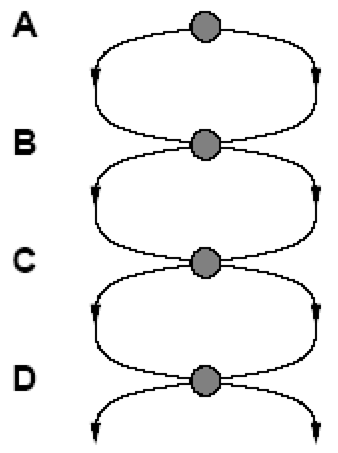
- Faster than BFS even though IDS generates repeated states
  - BFS generates nodes up to level d+1
  - IDS only generates nodes up to level d
- In general, iterative deepening search is the preferred uninformed search method when there is a large search space and the depth of the solution is not known

Avoiding Repeated States

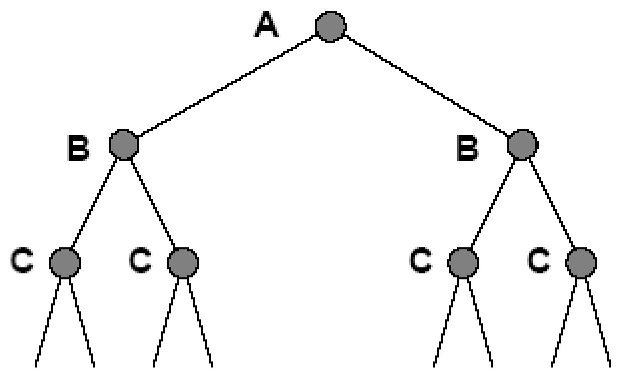
- Complication of wasting time by expanding states that have already been encountered and expanded before
  - Failure to detect repeated states can turn a linear problem into an exponential one

- Sometimes, repeated states are unavoidable
  - Problems where the actions are reversable
    - Route finding
    - Sliding blocks puzzles

Avoiding Repeated States

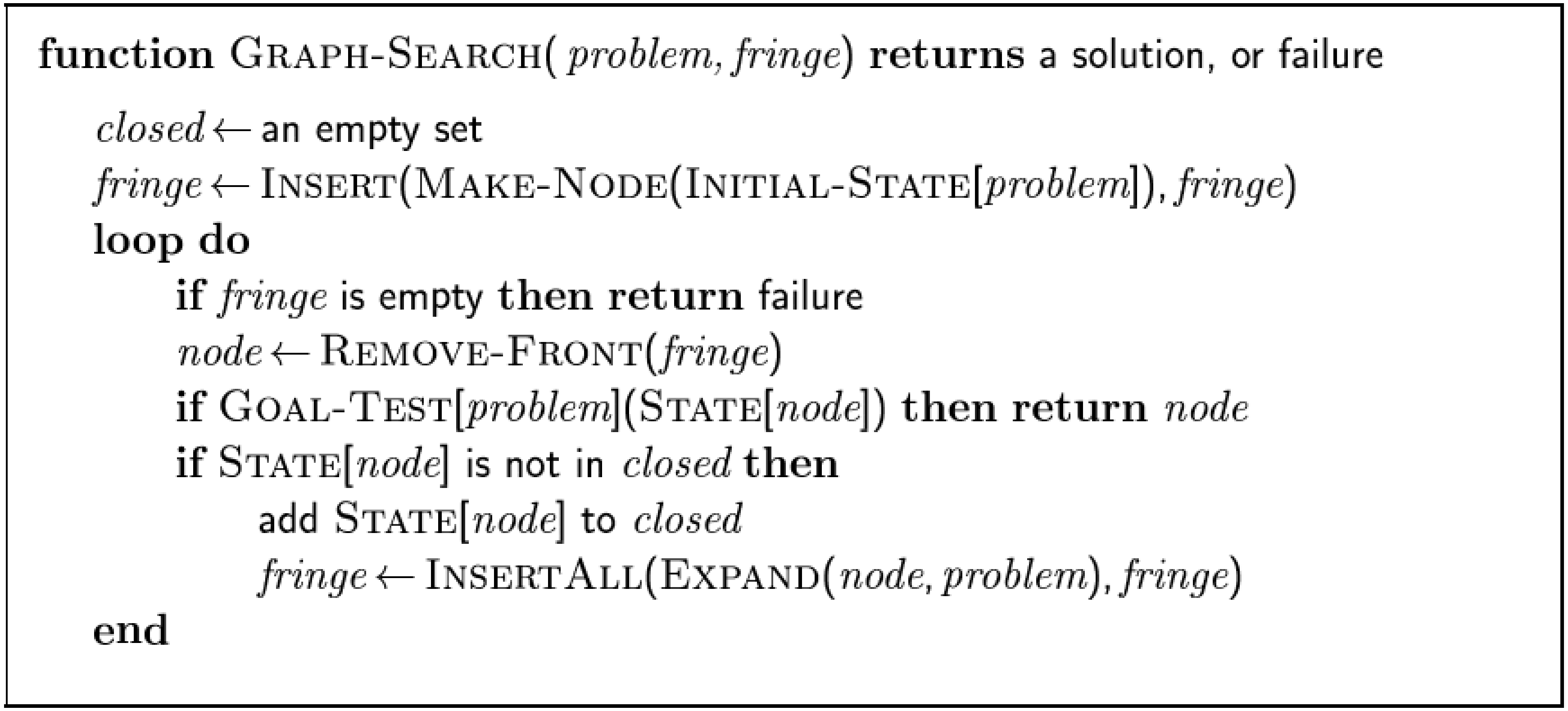


State Space



Search Tree

Avoiding Repeated States



```
function GRAPH-SEARCH( problem, fringe) returns a solution, or failure

    closed ← an empty set
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if fringe is empty then return failure
        node ← REMOVE-FRONT(fringe)
        if GOAL-TEST[problem](STATE[node]) then return node
        if STATE[node] is not in closed then
            add STATE[node] to closed
            fringe ← INSERTALL(EXPAND(node, problem), fringe)
    end
```

DFID - summary

- First do DFS to depth 0 (i.e., treat start node as having no successors), then, if no solution found, do DFS to depth 1, etc.
- until solution found do
-     DFS with depth cutoff c
-     c = c+1
- Complete
- Optimal/Admissible if all operators have the same cost. Otherwise, not optimal but guarantees finding solution of shortest length (like BFS).
- Time complexity seems worse than BFS or DFS because nodes near the top of the search tree are generated multiple times, but because almost all of the nodes are near the bottom of a tree, the worst case time complexity is still exponential, O(bd).

Depth-First Iterative Deepening

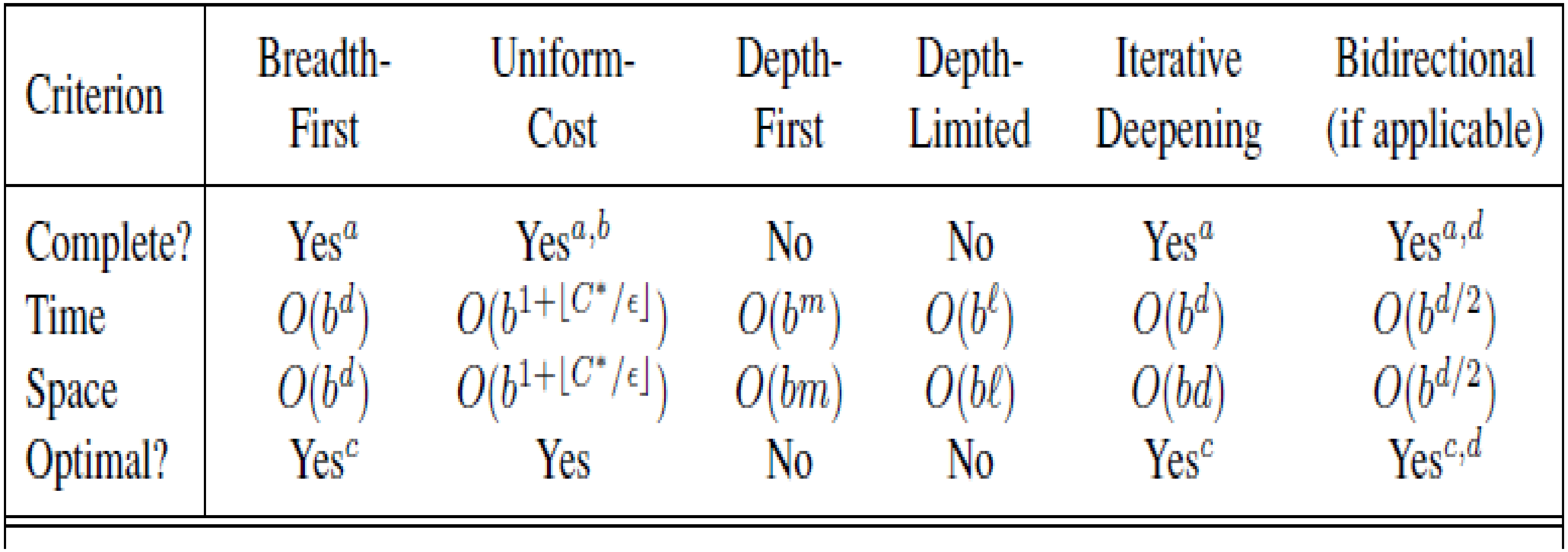
- If branching factor is b and solution is at depth d, then nodes at depth d are generated once, nodes at depth d-1 are generated twice, etc.
  - IDS : (d) b + (d-1) b2 + … + (2) b(d-1) + bd = O(bd).
  - If b=4, then worst case is 1.78 * 4d, i.e., 78% more nodes searched than exist at depth d (in the worst case).
- However, let's compare this to the time spent on BFS:
  - BFS : b + b2 + … + bd + (b(d+1) – b) = O(bd).
  - Same time complexity of O(bd), but BFS expands some nodes at depth d+1, which can make a HUGE difference:
    - With b = 10, d = 5,
    - BFS: 10 + 100 + 1,000 + 10,000 + 100,000 + 999,990 = 1,111,100
    - IDS: 50 + 400 + 3,000 + 20,000 + 100,000 = 123,450
- IDS can actually be quicker in-practice than BFS, even though it regenerates early states.

Depth-First Iterative Deepening

- Exponential time complexity, O(bd), like BFS
- Linear space complexity, O(bd), like DFS

- Has advantage of BFS (i.e., completeness) and also advantages of DFS (i.e., limited space and finds longer paths more quickly)
- Generally preferred for large state spaces where solution depth is unknown

Comparing uninformed search strategies



Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[a]	Yes[a,b]	No	No	Yes[a]	Yes[a,d]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes[c]	Yes	No	No	Yes[c]	Yes[c,d]

Evaluation of tree-search strategies: b is the branching factor; d is the depth of the shallowest solution; m is the maximum depth of the search tree; l is the depth limit.
Superscript caveats are as follows: a complete if b is finite; b complete if step costs ≥ for positive ; c optimal if step costs are all identical; d if both directions use breadth-first search.