UNIT I :

Introduction: What is AI, the foundations of AI, The state of the art Chapter I AI Textbook

Intelligent agents: Agents and environments, good behaviour, concept of rationality, nature of environments, structure of agents. Chapter 2 AI Textbook 03

Problem-solving: Problem-solving agents, Example problems, Searching for Solutions, Uninformed Search Strategies: Breadth First search, Depth First Search. Chapter 3 AI Textbook.

Q) Define AI. Describe the organization of AI Definition.

John McCarthy in mid-1950'scoined the term ―Artificial Intelligence‖ which he would define as ―the science and engineering of making intelligent machines‖ AI is about teaching the machines to learn, to act, and think as humans would do. We can organize AI definition into 4 categories:

- The definitions on top are concerned with thought processes and reasoning, whereas the ones on the bottom address behaviour.

- The definitions on the left measure success in terms of conformity to human performance whereas the ones on the right measure against an ideal performance measure called rationality.

- A system is rational if it does the "right thing," given what it knows.

- Historically, all four approaches to AI have been followed, each by different people with different methods.

- A human-cantered approach must be in part an empirical science, involving observations and hypotheses about human behaviour.

- A rationalist's approach involves a combination of mathematics and engineering. The various groups have both disparaged and helped each other. Let us look at the four approaches in more detail.

Thinking Humanly	**Thinking Rationally**
―The exciting new effort to make computers think machines with minds, in the full and literal sense. (Haugeland, 1985) ― [The automation of] activities that we associate with human thinking, activities such as decision-making, problem solving, learning. (Bellman, 1978)	―The study of mental faculties through the use of computational models. (Charniak and McDermott, 1985) ―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) ―The study of how to make computers do things at which, at the moment, people are better. (Rich and Knight, 1991)	―Computational Intelligence is the study of the design of intelligent agents. (Pooleet al., 1998) ―AI is concerned with intelligent behaviour in artifacts. (Nilsson, 1998

Acting humanly: The Turing Test approach:

The Turing Test, proposed by Alan Turing (1950), was designed to provide a satisfactory operational definition of intelligence. A computer passes the test if a human interrogator, after posing some written questions, cannot tell whether the written responses come from a person or from a computer. Chapter 26 discusses the details of the test and whether a computer would really be intelligent if it passed. For now, we note that programming a computer to pass a rigorously applied test provides plenty to work on. The computer would need to possess the following capabilities.

- **natural language processing** to enable it to communicate successfully in English.
- **knowledge representation** to store what it knows or hears.
- **automated reasoning** to use the stored information to answer questions and to draw new conclusions.
- **machine learning** to adapt to new circumstances and to detect and extrapolate patterns.

By distinguishing between human and rational behaviour, we are not suggesting that humans are necessarily "irrational" in the sense of "emotionally unstable" or "insane." One merely need note that we are not perfect: not all chess players are grandmasters; and, unfortunately, not everyone gets an A on the exam. Some systematic errors in human reasoning are catalogued by Kahneman et al. (1982).

Turing's test deliberately avoided direct physical interaction between the interrogator and the computer because physical simulation of a person is unnecessary for intelligence. However, the so-called total Turing Test includes a video signal so that the interrogator can test the subject's perceptual abilities, as well as the opportunity for the interrogator to pass physical objects "through the hatch." To pass the total Turing Test, the computer will need.

- **computer vision** to perceive objects, and
- **robotics to manipulate** objects and move about.

These six disciplines compose most of AI, and Turing deserves credit for designing a test that remains relevant 60 years later. Yet AI researchers have devoted little effort to passing the Turing Test, believing that it is more important to study the underlying principles of intelligence than to duplicate an exemplar. The quest for "artificial flight" succeeded when the Wright brothers and others stopped imitating birds and started using wind tunnels and learning about aerodynamics. Aeronautical engineering texts do not define the goal of their field as making "machines that fly so exactly like pigeons that they can fool even other pigeons."

Thinking humanly: The cognitive modelling approach:

If we are going to say that a given program thinks like a human, we must have some way of determining how humans think. We need to get inside the actual workings of human minds. There are three ways to do this: through introspection—trying to catch our own thoughts as they go by; through psychological experiments—observing a person in action; and through brain imaging—observing the brain in action. Once we have a sufficiently precise theory of the mind, it becomes possible to express the theory as a computer program. If the program's input–output behavior matches corresponding human behavior, that is evidence that some of the program's mechanisms could also be operating in humans. For example, Allen Newell and Herbert Simon, who developed GPS, the "General Problem Solver" (Newell and Simon, 1961), were not content merely to have their program solve problems correctly. They were more concerned with comparing the trace of its reasoning steps to traces of human subjects solving the same problems. The interdisciplinary field of cognitive science brings together computer models from AI and experimental techniques from psychology to construct precise and testable theories of the human mind.

Cognitive science is a fascinating field in itself, worthy of several textbooks and at least one encyclopedia (Wilson and Keil, 1999). We will occasionally comment on similarities or differences between AI techniques and human cognition. Real cognitive science, however, is necessarily based on experimental investigation of actual humans or animals. We will leave that for other books, as we assume the reader has only a computer for experimentation. In the early days of AI there was often confusion between the approaches: an author would argue that an algorithm performs well on a task and that it is therefore a good model of human performance, or vice versa. Modern authors separate the two kinds of claims; this distinction has allowed both AI and cognitive science to develop more rapidly. The two fields continue to fertilize each other, most notably in computer vision, which incorporates neurophysiological evidence into computational models.

Thinking rationally: The "laws of thought" approach:

The Greek philosopher Aristotle was one of the first to attempt to codify "right thinking," that is, irrefutable reasoning processes. His syllogisms provided patterns for argument structures that always yielded correct conclusions when given correct premises—for example, "Socrates is a man; all men are mortal; therefore, Socrates is mortal." These laws of thought were supposed to govern the operation of the mind; their study initiated the field called logic.

Logicians in the 19th century developed a precise notation for statements about all kinds of objects in the world and the relations among them. (Contrast this with ordinary arithmetic notation, which provides only for statements about numbers.) By 1965, programs existed that could, in principle, solve any solvable problem described in logical notation. (Although if no solution exists, the program might loop forever.) The so-called logicist tradition within artificial intelligence hopes to build on such programs to create intelligent systems.

There are two main obstacles to this approach. First, it is not easy to take informal knowledge and state it in the formal terms required by logical notation, particularly when the knowledge is less than 100% certain. Second, there is a big difference between solving a problem "in principle" and solving it in practice. Even problems with just a few hundred facts can exhaust the computational resources of any computer unless it has some guidance as to which reasoning steps to try first. Although both of these obstacles apply to any attempt to build computational reasoning systems, they appeared first in the logicist tradition.

Acting rationally: The rational agent approach:

An agent is just something that acts (agent comes from the Latin agere, to do). Of course, all computer programs do something, but computer agents are expected to do more: operate autonomously, perceive their environment, persist over a prolonged time period, adapt to change, and create and pursue goals. A rational agent is one that acts so as to achieve the best outcome or, when there is uncertainty, the best expected outcome. In the "laws of thought" approach to AI, the emphasis was on correct inferences. Making correct inferences is sometimes part of being a rational agent, because one way to act rationally is to reason logically to the conclusion that a given action will achieve one's goals and then to act on that conclusion. On the other hand, correct inference is not all of rationality; in some situations, there is no provably correct thing to do, but something must still be done. There are also ways of acting rationally that cannot be said to involve inference. For example, recoiling from a hot stove is a reflex action that is usually more successful than a slower action taken after careful deliberation. All the skills needed for the Turing Test also allow an agent to act rationally. Knowledge representation and reasoning enable agents to reach good decisions. We need to be able to generate comprehensible sentences in natural language to get by in a complex society. We need learning not only for erudition, but also because it improves our ability to generate effective behavior. The rational-agent approach has two advantages over the other approaches. First, it is more general than the "laws of thought" approach because correct inference is just one of several possible mechanisms for achieving rationality. Second, it is

more amenable to scientific development than are approaches based on human behavior or human thought. The standard of rationality is mathematically well defined and completely general, and can be "unpacked" to generate agent designs that provably achieve it. Human behavior, on the other hand, is well adapted for one specific environment and is defined by, well, the sum total of all the things that humans do. This book therefore concentrates on general principles of rational agents and on components for constructing them. We will see that despite the apparent simplicity with which the problem can be stated, an enormous variety of issues come up when we try to solve it. Chapter 2 outlines some of these issues in more detail. One important point to keep in mind: We will see before too long that achieving perfect rationality—always doing the right thing—is not feasible in complicated environments. The computational demands are just too high. For most of the book, however, we will adopt the working hypothesis that perfect rationality is a good starting point for analysis. It simplifies the problem and provides the appropriate setting for most of the foundational material in the field. Chapters 5 and 17 deal explicitly with the issue of limited rationality—acting appropriately when there is not enough time to do all the computations one might like.

Beneficial machines

The standard model has been a useful guide for AI research since its inception, but it is probably not the right model in the long run. The reason is that the standard model assumes that we will supply a fully specified objective to the machine.

The problem of achieving agreement between our true preferences and the objective we put into the machine is called the *value alignment problem**: the values or objectives put into the machine must be aligned with those of the human.

We don't want machines that are intelligent in the sense of pursuing *their* objectives; we want them to pursue *our* objectives. Ultimately, we want agents that are **provably beneficial** to humans.

The Foundations of Artificial Intelligence:

Philosophy:

- Can formal rules be used to draw valid conclusions?
- How does the mind arise from a physical brain?
- Where does knowledge come from?
- How does knowledge lead to action?

Descartes was a proponent of **dualism**. He held that there is a part of the human mind (or soul or spirit) that is outside of nature, exempt from physical laws. Animals, on the other hand, did not possess this dual quality; they could be treated as machines.

An alternative to dualism is **materialism**, which holds that the brain's operation according to the laws of physics constitutes the mind. Free will is simply the way that the perception of available choices appears to the choosing entity.

The **empiricism** is characterized by a dictum of John Locke (1632–1704): "Nothing is in the understanding, which was not first in the senses."

The **logical positivism** holds that all knowledge can be characterized by logical theories connected, ultimately, to **observation sentences** that correspond to sensory inputs; thus logical positivism combines rationalism and empiricism.

The **confirmation theory** attempted to analyze the acquisition of knowledge from experience by quantifying the degree of belief that should be assigned to logical sentences based on their connection to observations that confirm or disconfirm them.

Utilitarianism: that rational decision making based on maximizing utility should apply to all spheres of human activity, including public policy decisions made on behalf of many individuals. Utilitarianism is a specific kind of **consequentialism**: the idea that what is right and wrong is determined by the expected outcomes of an action.

In contrast, a theory of rule-based or **deontological ethics**, in which "doing the right thing" is determined not by outcomes but by universal social laws that govern allowable actions, such as "don't lie" or "don't kill." Many modern AI systems adopt exactly this approach.

Mathematics:

- What are the formal rules to draw valid conclusions?
- What can be computed?
- How do we reason with uncertain information?

The theory of **probability** can be seen as generalizing logic to situations with uncertain information—a consideration of great importance for AI. The formalization of probability, combined with the availability of data, led to the emergence of **statistics** as a field.

The history of computation is as old as the history of numbers, but the first nontrivial **algorithm** is thought to be Euclid's algorithm for computing greatest common divisors. The **incompleteness theorem** showed that in any formal theory as strong as Peano arithmetic (the elementary theory of natural numbers), there are necessarily true statements that have no proof within the theory.

Alan Turing tried to characterize exactly which functions *are* **computable** —capable of being computed by an effective procedure. For example, no machine can tell *in general* whether a given program will return an answer on a given input or run forever.

Although computability is important to an understanding of computation, the notion of **tractability** has had an even greater impact on AI. A problem is called intractable if the time required to solve instances of the problem grows exponentially with the size of the instances.

The theory of **NP-completeness** provides a basis for analysing the tractability of problems: any problem class to which the class of NP-complete problems can be reduced is likely to be intractable.

Economics:

- How should we make decisions in accordance with our preferences?
- How should we do this when others may not go along?
- How should we do this when the payoff may be far in the future?

Decision theory, which combines probability theory with utility theory, provides a formal and complete framework for individual decisions (economic or otherwise) made under uncertainty—that is, in cases where probabilistic descriptions appropriately capture the decision maker's environment.

Models based on **satisficing**—making decisions that are "good enough," rather than laboriously calculating an optimal decision—gave a better description of actual human behaviour.

Neuroscience:

- How do brains process information?

Neuroscience is the study of the nervous system, particularly the brain. A collection of simple cells can lead to thought, action, and consciousness.

Even with a computer of virtually unlimited capacity, we still require further conceptual breakthroughs in our understanding of intelligence.

Psychology:

- How do humans and animals think and act?

Wundt insisted on carefully controlled experiments in which his workers would perform a perceptual or associative task while introspecting on their thought processes. The **behaviourism** movement rejected any theory involving mental processes on the grounds that introspection could not provide reliable evidence.

Cognitive psychology views the brain as an information-processing device. Three key steps of a knowledge-based agent: (1) the stimulus must be translated into an internal representation, (2) the representation is manipulated by cognitive processes to derive new internal representations, and (3) these are in turn retranslated back into action.

Intelligence augmentation states that computers should augment human abilities rather than automate away human tasks.

Computer engineering:

- How can we build an efficient computer?

Moore's law states that performance doubled every 18 months. **Quantum computing** holds out the promise of far greater accelerations for some important subclasses of AI algorithms.

Control theory and cybernetics:

- How can artifacts operate under their own control?

Modern control theory, especially the branch known as stochastic optimal control, has as its goal the design of systems that maximize a **cost function** over time.

Linguistics:

- How does language relate to thought?

Modern linguistics and AI, then, were "born" at about the same time, and grew up together, intersecting in a hybrid field called **computational linguistics** or **natural language processing**.

The History of Artificial Intelligence:

The inception of artificial intelligence (1943-1956) : The first work that is now generally recognized as AI was done by Warren McCulloch and Walter Pitts (1943).

Early enthusiasm, great expectations (1952-1969): The intellectual establishment of the 1950s, by and large, preferred to believe that "a machine can never do X". John McCarthy referred to this period as the "Look, Ma, no hands!" era.

A dose of reality (1966-1973): In almost all cases these early systems failed on more difficult problems. The illusion of unlimited computational power was not confined to problem-solving programs.

Expert systems (1969-1986): The picture of problem solving that had arisen during the first decade of AI research was of a general-purpose search mechanism trying to string together elementary reasoning steps to find complete solutions. Such approaches have been called **weak methods**. The alternative to weak methods is to use more powerful, domain-specific knowledge that allows larger reasoning steps and can more easily handle typically occurring cases in narrow areas of expertise.

The return of neural networks (1986-present): In the mid-1980s at least four different groups reinvented the **back-propagation** learning algorithm first developed in the early 1960s.

Probabilistic reasoning and machine learning (1987-present): In the 1980s, approaches using **hidden Markov models** (HMMs) came to dominate the area. Pearl's development of **Bayesian networks** yielded a rigorous and efficient formalism for representing uncertain knowledge as well as practical algorithms for probabilistic reasoning.

Big data (2001-present): Remarkable advances in computing power and the creation of the World Wide Web have facilitated the creation of very large data sets—a phenomenon sometimes known as **big data**.

Deep learning (2011-present) The term **deep learning** refers to machine learning using multiple layers of simple, adjustable computing elements. Experiments were carried out with such networks as far back as the 1970s, and in the form of **convolutional neural networks** they found some success in handwritten digit recognition in the 1990s.

The Sate of the Art:

What can AI do today? A concise answer is difficult because there are so many activities in so many subfields. Here we sample a few applications; others appear throughout the book.

Robotic vehicles:

A driverless robotic car named STANLEY sped through the rough terrain of the Mojave dessert at 22 mph, finishing the 132-mile course first to win the 2005 DARPA Grand Challenge. STANLEY is a Volkswagen Touareg outfitted with cameras, radar, and laser rangefinders to sense the environment and onboard software to command the steering, braking, and acceleration (Thrun, 2006). The following year CMU's BOSS won the Urban Challenge, safely driving in traffic through the streets of a closed Air Force base, obeying traffic rules and avoiding pedestrians and other vehicles.

Speech recognition:

A traveller calling United Airlines to book a flight can have the entire conversation guided by an automated speech recognition and dialog management system.

Autonomous planning and scheduling: A hundred million miles from Earth, NASA's Remote Agent program became the first on-board autonomous planning program to control the scheduling of operations for a spacecraft (Jonsson et al., 2000). REMOTE AGENT generated plans from high-level goals specified from the ground and monitored the execution of those plans—detecting, diagnosing, and recovering from problems as they occurred. Successor program MAPGEN (Al-Chang et al., 2004) plans the daily operations for NASA's Mars Exploration Rovers, and MEXAR2 (Cesta et al., 2007) did mission planning—both logistics and science planning—for the European Space Agency's Mars Express mission in 2008.

Game playing: IBM's DEEP BLUE became the first computer program to defeat the world champion in a chess match when it bested Garry Kasparov by a score of 3.5 to 2.5 in an exhibition match (Goodman and Keene, 1997). Kasparov said that he felt a "new kind of intelligence" across the board from him. Newsweek magazine described the match as "The brain's last stand." The value of IBM's stock increased by $18 billion. Human champions studied Kasparov's loss and were able to draw a few matches in subsequent years, but the most recent human-computer matches have been won convincingly by the computer.

Spam fighting: Each day, learning algorithms classify over a billion messages as spam, saving the recipient from having to waste time deleting what, for many users, could comprise 80% or 90% of all messages, if not classified away by algorithms. Because the spammers are continually updating their tactics, it is difficult for a static programmed approach to keep up, and learning algorithms work best (Sahami et al., 1998; Goodman and Heckerman, 2004).

Logistics planning: During the Persian Gulf crisis of 1991, U.S. forces deployed a Dynamic Analysis and Replanning Tool, DART (Cross and Walker, 1994), to do automated logistics planning and scheduling for transportation. This involved up to 50,000 vehicles, cargo, and people at a time, and had to account for starting points, destinations, routes, and conflict resolution among all parameters. The AI planning techniques generated in hours a plan that would have taken weeks with older methods. The Defense Advanced Research Project Agency (DARPA) stated that this single application more than paid back DARPA's 30-year investment in AI.

Robotics: The iRobot Corporation has sold over two million Roomba robotic vacuum cleaners for home use. The company also deploys the more rugged PackBot to Iraq and Afghanistan, where it is used to handle hazardous materials, clear explosives, and identify the location of snipers.

Machine Translation: A computer program automatically translates from Arabic to English, allowing an English speaker to see the headline "Ardogan Confirms That Turkey Would Not Accept Any Pressure, Urging Them to Recognize Cyprus." The program uses a statistical model built from examples of Arabic-to-English translations and from examples of English text totaling two trillion words (Brants et al., 2007). None of the computer scientists on the team speak Arabic, but they do understand statistics and machine learning algorithms.

These are just a few examples of artificial intelligence systems that exist today. Not magic or science fiction—but rather science, engineering, and mathematics, to which this book provides an introduction.

Intelligent agents: Agents and environments, good behaviour, concept of rationality, nature of environments, structure of agents.

Intelligent Agents:

Agents and Environments:

An **agent** is anything that can be viewed as perceiving its **environment** through **sensors** and acting upon that environment through **actuators**.

- An **agent** is anything that can be ENVIRONMENT viewed as perceiving its **environment** through **sensors** and SENSOR acting upon that environment through **actuators**.
- **ACTUATOR**: A human agent has eyes, ears, and other organs for sensors and hands, legs, vocal tract, and so on for actuators.
- **A robotic agent:** cameras and infrared range finders for sensors and various motors for actuators.
- A software agent receives keystrokes, file contents, and network packets as sensory inputs and acts on the environment by displaying on the screen, writing files, and sending network packets.

This simple idea is illustrated in Figure, A human agent has eyes, ears, and other organs for sensors and hands, legs, vocal tract, and so on for actuators. A robotic agent might have cameras and infrared range finders for sensors and various motors for actuators. A software agent receives keystrokes, file contents, and network packets as sensory inputs and acts on the environment by displaying on the screen, writing files, and sending network packets.

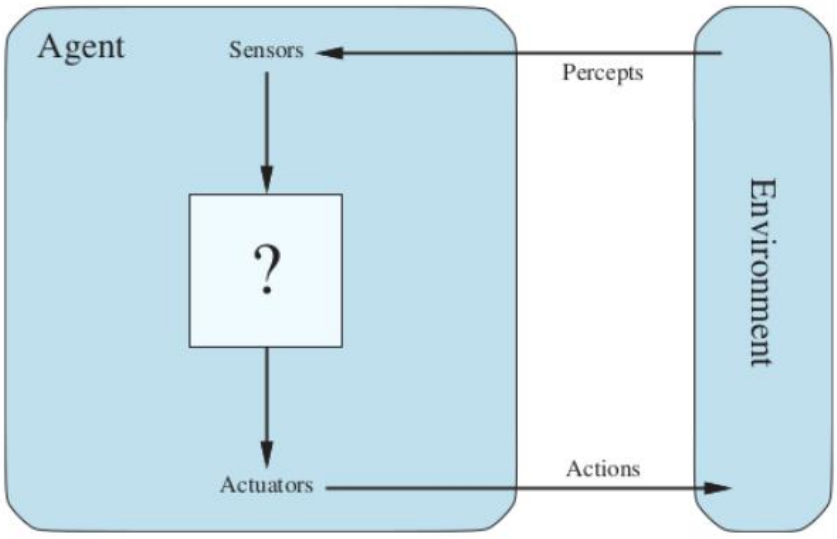


Figure 2.1 Agents interact with environments through sensors and actuators.

We use the term **percept** to refer to the content an agent's sensors are perceiving. An agent's **percept sequence** is the complete history of everything the agent has ever perceived. In general, an agent's choice of action at any given instant can depend on its built-in knowledge and on the entire percept sequence observed to date, but not on anything it hasn't perceived. Mathematically speaking, we say that an agent's behaviour is described by the **agent function** that maps any given percept sequence to an action.

Internally, the agent function for an artificial agent will be implemented by an **agent program**. It is important to keep these two ideas distinct. The agent function is an abstract mathematical description; the agent program is a concrete implementation, running within some physical system.

Vacuum-Cleaner World:

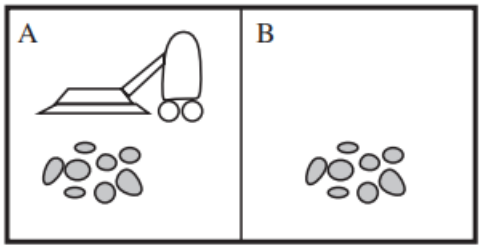


Figure 2.2 A vacuum-cleaner world with just two locations.

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
⋮	⋮
[A, Clean], [A, Clean], [A, Clean]	Right
[A, Clean], [A, Clean], [A, Dirty]	Suck
⋮	⋮

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

To illustrate these ideas, we use a very simple example—the vacuum-cleaner world shown in Figure 2.2. This world is so simple that we can describe everything that happens; it's also a made-up world, so we can invent many variations. This particular world has just two locations: squares A and B. The vacuum agent perceives which square it is in and whether there is dirt in the square. It can choose to move left, move right, suck up the dirt, or do nothing. One very simple agent function is the following: if the current square is dirty, then suck; otherwise, move to the other square. A partial tabulation of this agent function is shown in Figure 2.3 and an agent program that implements it appears in Figure 2.8 on page 48. Looking at Figure 2.3, we see that various vacuum-world agents can be defined simply by filling in the right-hand column in various ways. The obvious question, then, is this: What is the right way to fill out the table? In other words, what makes an agent good or bad, intelligent or stupid? We answer these questions in the next section.

Good Behaviour, Concept of Rationality:

- Rational agent does "the right thing"
  The action that leads to the best outcome under the given circumstances
- An agent function maps percept sequences to actions
  Abstract mathematical description
- An agent program is a concrete implementation of the respective function
  It runs on a specific agent architecture ("platform") on physical devices.
- Problems:
- What is "the right thing", How do you measure the "best outcome"

A rational agent **is one that does the right thing.**

Performance measures:

Moral philosophy has developed several different notions of the "right thing," but AI has generally stuck to one notion called **consequentialism**: we evaluate an agent's behaviour by its consequences.

This notion of desirability is captured by a **performance measure** that evaluates any given sequence of environment states.

Rationality

What is rational at any given time depends on four things:

- The performance measure that defines the criterion of success.
- The agent's prior knowledge of the environment.
- The actions that the agent can perform.
- The agent's percept sequence to date.

Definition of a 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 the agent has.

Rationality and omniscience:

- An omniscient agent knows the *actual* outcome of its actions and can act accordingly; but omniscience is impossible in reality.
- **Rationality** maximizes expected performance, while **perfection** maximizes *actual* performance.
- **Omniscience**
- An omniscient agent knows the actual outcome of its actions and can act accordingly; but omniscience is impossible in reality
- **learning**
- **Rational agent:**
- Learn and gather information from what it perceives.
- The agent's initial configuration could reflect some prior knowledge of the environment, but as the agent gains experience this may be modified and augmented.
- There are extreme cases in which the environment is completely known *a* **priori.**
- **autonomy**
- A rational agent should be autonomous—it should learn what it can to compensate for partial or incorrect prior knowledge.
- **task environment**
- Task environments, which are essentially the "problems" to which rational agents are the "solutions.

Omniscience, learning, and autonomy:

We need to be careful to distinguish between rationality and **omniscience**. An omniscient agent knows the actual outcome of its actions and can act accordingly; but omniscience is impossible in reality.
Rationality maximizes expected performance, while perfection maximizes actual performance.
Doing actions in order to modify future precepts—sometimes called **information gathering**. Our definition requires a rational agent not only to gather information but also to **learn** as much as possible from what it perceives.
To the extent that an agent relies on the prior knowledge of its designer rather than on its own precepts and learning processes, we say that the agent lacks **autonomy**. A rational agent should be autonomous—it should learn what it can to compensate for partial or incorrect prior knowledge.

Task environment

The Nature of Environments:

Now that we have a definition of rationality, we are almost ready to think about building rational agents. First, however, we must think about task environments, which are essentially the "problems" to which rational agents are the "solutions." We begin by showing how to specify a task environment, illustrating the process with a number of examples. We then show that task environments come in a variety of Flavors. The flavour of the task environment directly affects the appropriate design for the agent program.

Specifying the task environment:

In our discussion of the rationality of the simple vacuum-cleaner agent, we had to specify the performance measure, the environment, and the agent's actuators and sensors. We group all these under the heading of the task environment.
For the acronymically minded, we call is the PEAS (Performance, Environment, Actuators, Sensors) description. In designing an agent, the first step must always be to specify the task environment as fully as possible.
The vacuum world was a simple example; let us consider a more complex problem:
An automated taxi driver. We should point out, before the reader becomes alarmed, that a fully automated taxi is currently somewhat beyond the capabilities of existing technology. (Page 28 describes an existing driving robot.) The full driving task is extremely open-ended. There is no limit to the novel combinations of circumstances that can arise—another reason we chose it as a focus for discussion. Figure 2.4 summarizes the PEAS description for the taxi's task environment. We discuss each element in more detail in the following paragraphs.

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.

1. **performance measure:** getting to the correct destination; minimizing fuel consumption, and wear and tear; minimizing the trip time or cost; minimizing violations of traffic, laws and disturbances to other drivers; maximizing safety and passenger comfort; maximizing profits.
2. **Environment:** roads, pedestrians, stray animals, road works, police cars, puddles and potholes, interact with potential and actual passengers.

3. **Actuators:** control over the engine through the accelerator and control over steering and braking. Output to a display screen or voice synthesizer to talk back to the passengers, and perhaps some way to communicate with other vehicles, politely or otherwise.
4. **sensors:** one or more controllable video cameras, infrared or sonar sensors to detect distances to other cars and obstacles. A speedometer, and to control the vehicle properly, especially on curves, it should have an accelerometer. GPS, Fuel sensors. Finally, it will need a keyboard or microphone for the passenger to request a destination.
5. **software agents** (or software robots or **softbots**) exist in rich, unlimited domains.
6. Imagine a softbot Web site operator designed to scan Internet news sources and show the interesting items to its users, while selling advertising space to generate revenue.

Properties of task environments:

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
Satellite image analysis system	Correct image categorization	Downlink from orbiting satellite	Display of scene categorization	Color pixel arrays
Part-picking robot	Percentage of parts in correct bins	Conveyor belt with parts; bins	Jointed arm and hand	Camera, joint angle sensors
Refinery controller	Purity, yield, safety	Refinery, operators	Valves, pumps, heaters, displays	Temperature, pressure, chemical sensors
Interactive English tutor	Student's score on test	Set of students, testing agency	Display of exercises, suggestions, corrections	Keyboard entry

Figure 2.5 Examples of agent types and their PEAS descriptions.

1.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.
- All aspects that are relevant to the choice of action;
- agent need not maintain any internal state
- partially observable: because of noisy and inaccurate sensors or because parts of the state are simply missing from the sensor data.
- 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.
- If the agent has no sensors at all then the environment is unobservable.

2.Single agent vs. multiagent:

- an agent solving a crossword puzzle by itself is clearly in a single-agent environment
- agent playing chess is in a two-agent environment.

3.Deterministic vs. stochastic

- If the next state of the environment is completely determined by the current state and the action executed by the agent.
- otherwise, it is stochastic.
- If partially observable-→ could be stochastic.
- The environment is uncertain if it is not fully observable or not deterministic.
- Nondeterministic environment is one in which actions are characterized by their possible outcomes, but no probabilities are attached to them.

4.Episodic vs. sequential

- The agent's experience is divided into atomic episodes.
- In each episode the agent receives a percept and then performs a single action.
- Next episode does not depend on the actions taken in previous episodes. Many classification tasks are episodic
- sequential:
- current decision could affect all future decisions.
- Ex: Chess and taxi driving are sequential: in both cases, short-term actions can have long-term consequences.
- Episodic environments are much simpler than sequential environments because the agent does not need to think ahead.

5.Static vs. dynamic

- The environment and agent changes - dynamic otherwise, 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.
- Ex: Crossword puzzles are static.
- Dynamic environments are continuously asking the agent what it wants to do; if it hasn't decided yet, that counts as deciding to do nothing.
- Ex: Taxi driving
- If the environment itself does not change with the passage of time but the agent's performance score does – semi dynamic.
- Ex: Chess, when played with a clock, is semi dynamic

6.Discrete vs. continuous:

- *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.

7.Known vs. unknown:

- Known: outcomes or outcome probabilities for all actions are given.
- Unknown: the agent will have to learn how it works in order to make good decisions.
- It is quite possible for a *known* environment to be *partially* observable—
- Ex: solitaire card games: I know the rules but am still unable to see the cards that have not yet been turned over.
- *Unknown* environment: can be fully observable—in a new video game, the screen may show the entire game state but I still don't know what the buttons do until I try them.

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.

The Structure of Agents:

The job of AI is to design an **agent program** that implements the agent function—the mapping from percepts to actions. We assume this program will run on some sort of computing device with physical sensors and actuators—we call this the **agent architecture**:

Agent = Architecture + Program

Agent programs:

- The agent program takes just the current percept as input;
- If the agent's actions need to depend on the entire percept sequence, the agent will have to remember the percepts.
- Let **P** be the set of possible percepts and let T be the lifetime of the agent (the total number of percepts it will receive).
- The lookup table will contain entries.

$$\sum_{t=1}^{T} |P|^t$$

```
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.

Ex: Consider the automated taxi: the visual input from a single camera comes in at the rate of roughly 27 megabytes per second (30 frames per second, 640×480 pixels with 24bits of color information). This gives a lookup table with over $10^{250,000,000,000}$ entries for an hour's driving.

The daunting size of these tables means that (a) no physical agent in this universe will have the space to store the table; (b) the designer would not have time to create the table; and (c) no agent could ever learn all the right table entries from its experience.
The key challenge for AI is to find out how to write programs that, to the extent possible, produce rational behaviour from a smallish program rather than from a vast table.

size of these tables:
a) Table entries are larger
b) Designer would not have time to create the table
c) All the right table entries cannot be learnt from its experience,
d) In a simple environment lack of guidance to fill the table entry is required.

TABLE-DRIVEN-AGENT does do what we want: it implements the desired agent function.
- Simple reflex agents;
- Model-based reflex agents;
- Goal-based agents; and
- Utility-based agents.

Simple reflex agents:
- simplest kind of agent.
- Select actions based on current percept, ignoring percept history

Ex: vacuum agent: its decision is based only on the current location and on whether that location contains dirt.
A more general and flexible approach is first to build a general-purpose interpreter for condition–action rules and then to create rule sets for specific task environments.

```
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-location vacuum environment. This program implements the agent function tabulated in Figure **??**.

```
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.

Actual Implementations: Using Logic Gates with A Boolean Circuit.

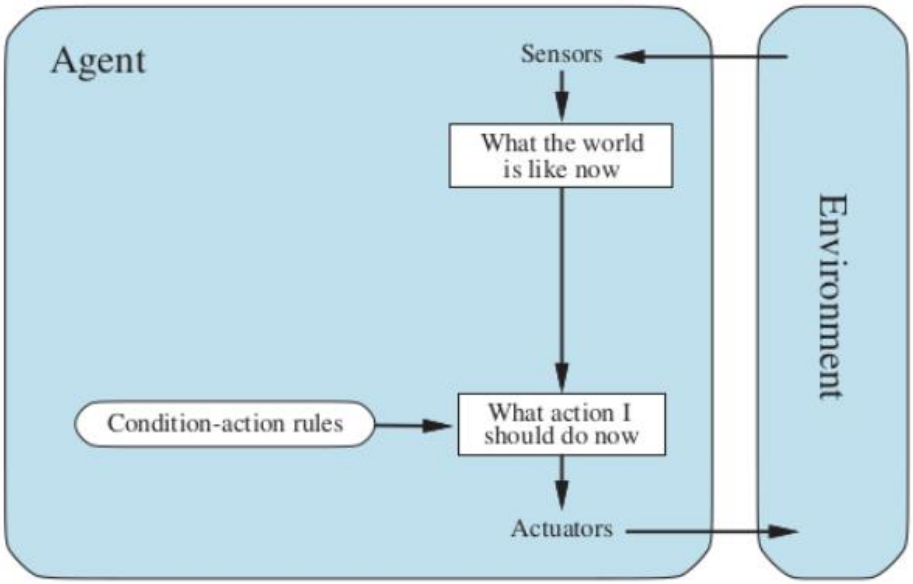


Figure 2.9 Schematic diagram of a simple reflex agent. We use rectangles to denote the current internal state of the agent's decision process, and ovals to represent the background information used in the process.

condition–action rule: automated taxi If the car in front brakes and its brake lights come on, then you should notice this and initiate braking. In other words, some processing is done on the visual input to establish the condition we call "The car in front is braking." Then, this triggers some established connection in the agent program to the action "initiate braking.
if car-in-front-is-braking then initiate-braking
condition–action rule: if car-in-front-is-braking then initiate-braking.
Note:

- ❖ rectangles to denote the current internal state of the agent's decision process,
- ❖ ovals to represent the background information used in the process.
- ❖ Rectangles: current internal state of the agent's decision process
- ❖ Ovals: background information used in the process.

Limitations:
1.Admirable property of being simple, but they turn out to be of limited intelligence.
Ex: only the current percept—that is, only if the environment is fully observable.
Suppose that a simple reflex vacuum agent is deprived of its location sensor and has only a dirt sensor. Such an agent
has just two possible precepts: [Dirty] and [Clean]. It can Suck in response to [Dirty]; what should it do in response to [Clean]? Moving Left fails (forever) if it happens to start in square A, and moving Right fails (forever) if it happens to start in square B. Infinite loops are often unavoidable for simple reflex agents operating in partially observable environments.

2.Escape from infinite loops is possible if the agent can randomize its actions.
For example, if the vacuum agent perceives [Clean], it might flip a coin to choose between Left and Right . It is easy to show that the agent will reach the other square in an average of two steps. Then, if that square is dirty, the agent will clean it and the task will be complete. Hence, a randomized simple reflex agent might outperform a deterministic simple reflex agent.

Model-based reflex agents:

- The solution to partial observability is to keep track of the part of the world it can't see now.
- Agent – Maintain internal state with percept history and thereby reflects at least some of the unobserved aspects of the current state.

Updating the internal states:

- First, some information about how the world evolves independently of the agent
- Ex: Overtaking car generally will be closer behind than it was a moment ago.
- Second, need some information about how the agent's own actions affect the world
- Ex: when the agent turns the steering wheel clockwise, the car turns to the right, or anticlockwise towards left.
- knowledge about "how the world work is called a model of the world. An agent that uses such a model is called a model-based agent.

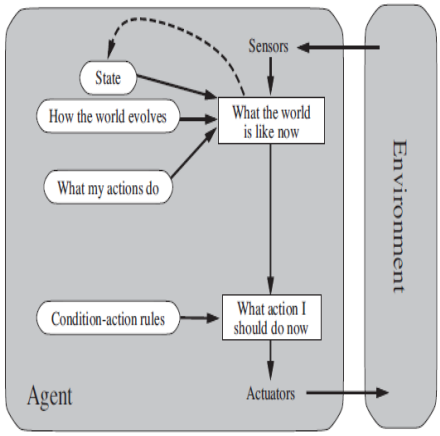


Figure 2.11 A model-based reflex agent.

```
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:

current state description, the GOAL agent needs some sort of goal information with situation desirable Ex: passenger's destination.

The agent program can combine this with the model actions that achieve the goal.

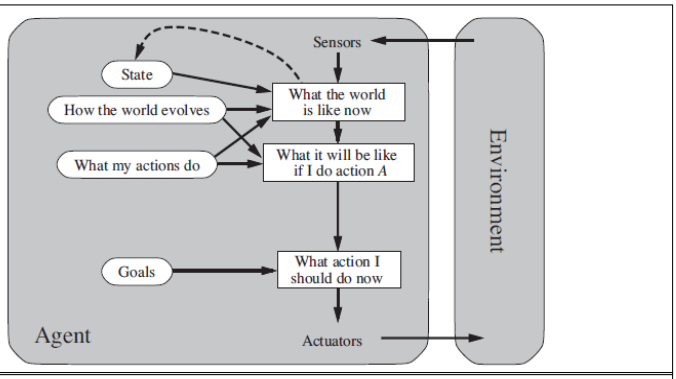


Figure 2.13 A model-based, goal-based agent. It keeps track of the world state as well as a set of goals it is trying to achieve, and chooses an action that will (eventually) lead to the achievement of its goals.

when goal satisfaction results immediately from a single action. Sometimes it will be more tricky—for example, when the agent has to consider long sequences of twists and turns in order to find a way to achieve the goal. Search and planning are the subfields of AI devoted to finding action sequences that achieve the agent's goals.

Ex: The reflex agent brakes when it sees brake lights. A goal-based agent, in principle, could reason that if the car in front has its brake lights on, it will slow down

Difference between goal based and reflex agents:

Goal based:

Less efficient, it is more flexible because the knowledge that supports its decisions is represented explicitly and can be modified.

The goal-based agent's behaviour can easily be changed to go to a different destination, simply by specifying that destination as the goal.

Reflex agent:

Need to rewrite many condition–action rules.

The reflex agent's rules for when to turn and when to go straight will work only for a single destination; they must all be replaced to go somewhere new.

Utility-based agents:

Goals- "happy" and "unhappy" states.

Because "happy" is not very scientific, economists and computer scientists use the term utility instead

An agent's utility function is performance measure.

If the internal utility function and the external performance measure are in agreement, then an agent that chooses actions to maximize its utility will be rational according to the external performance measure.

many advantages in terms of flexibility and learning.

goals are inadequate but a utility-based agent can still make rational decisions.

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, decision making under uncertainty

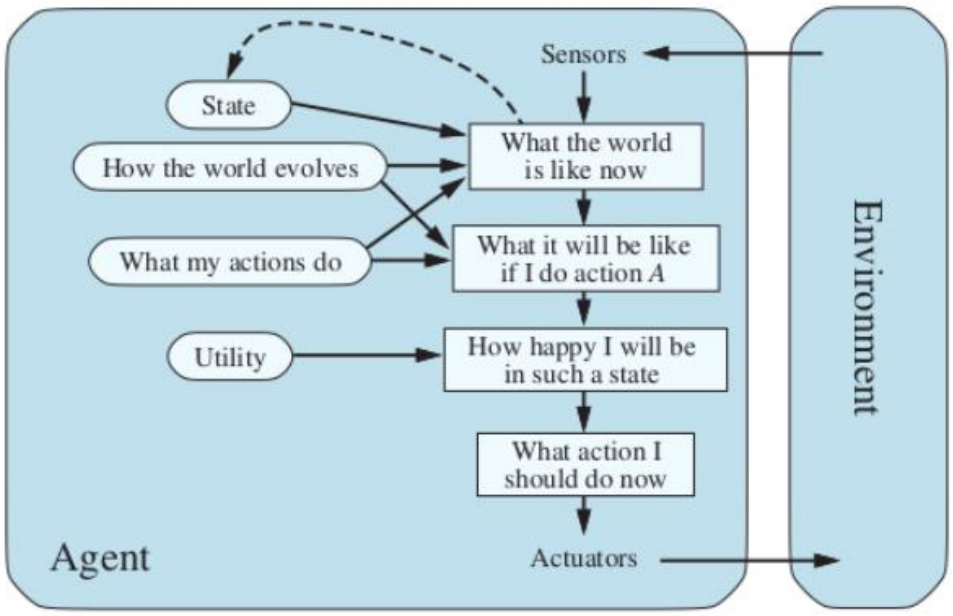


Figure 2.14 A model-based, utility-based agent. It uses a model of the world, along with a utility function that measures its preferences among states of the world. Then it chooses the action that leads to the best expected utility, where expected utility is computed by averaging over all possible outcome states, weighted by the probability of the outcome.

Learning agents:

how the agent programs come into being

1.Learning element- A learning agent can be divided into four conceptual components which is responsible for making improvements
2. Performance element- which is responsible for selecting external actions.
The performance element is what we have previously considered to be the entire agent: it takes in percepts and decides on actions.
3. critic: 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.
4. problem generator: The last component of the learning agent is the problem generator ,suggesting actions that will lead to new and informative experiences.

If the performance element had its way, it would keep doing the actions that are best, given what it knows. The performance standard distinguishes part of the incoming percept as a reward (or penalty) that provides direct feedback on the quality of the agent's behavior.

How the components of agent programs work:

What is the world like now?"
"What action should I do now?"
"What do my actions do?"
next question for a student of AI How on earth do these components work?"
representations along an axis of increasing complexity and expressive power—**atomic**, **factored**, and **structured**

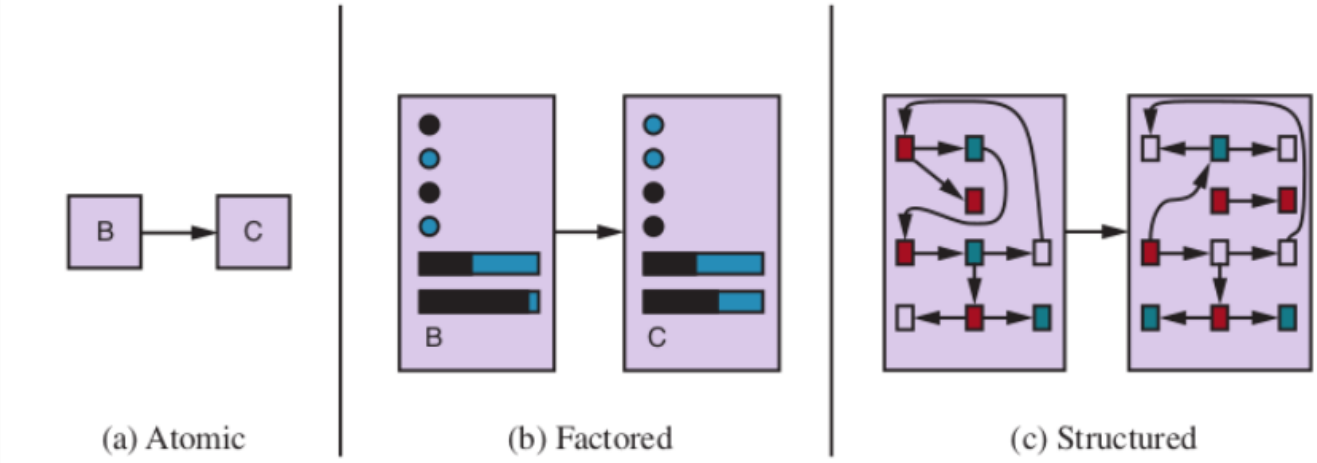


Figure 2.16 Three ways to represent states and the transitions between them. (a) Atomic representation: a state (such as B or C) is a black box with no internal structure; (b) Factored representation: a state consists of a vector of attribute values; values can be Boolean, real-valued, or one of a fixed set of symbols. (c) Structured representation: a state includes objects, each of which may have attributes of its own as well as relationships to other objects.

Roughly speaking, we can place the representations along an axis of increasing complexity and expressive power—atomic, factored, and structured.
In an **atomic representation** each state of the world is indivisible—it has no internal structure. The standard algorithms underlying search and game-playing, hidden Markov models, and Markov decision processes all work with atomic representations.
A **factored representation** splits up each state into a fixed set of **variables** or **attributes**, each of which can have a **value**. Many important areas of AI are based on factored representations, including constraint

satisfaction algorithms, propositional logic, planning, Bayesian networks, and various machine learning algorithms.
In a **Structured representation**, objects and their various and varying relationships can be described explicitly. Structured representations underlie relational databases and first-order logic, first-order probability models, and much of natural language understanding.
The axis along which atomic, factored, and structured representations lie is the axis of increasing **expressiveness**. Roughly speaking, a more expressive representation can capture, at least as concisely, everything a less expressive one can capture, plus some more.
Another axis for representation involves the mapping of concepts to locations in physical memory, whether in a computer or in a brain. If there is a one-to-one mapping between concepts and memory locations, we call that a **localist representation**. On the other hand, if the representation of a concept is spread over many memory locations, and each memory location is employed as part of the representation of multiple different concepts, we call that a **distributed representation**. Distributed representations are more robust against noise and information loss.

Problem-solving: Problem-solving agents, Example problems, Searching for Solutions, Uninformed Search Strategies: Breadth First search, Depth First Search. Chapter 3 AI Textbook.

Problem-solving agents:

A type of intelligent agent designed to address and solve complex problems or tasks in its environment

When the correct action to take is not immediately obvious, an agent may need to plan ahead: to consider a sequence of actions that form a path to a goal state. Such an agent is called a **problem-solving agent**, and the computational process it undertakes is called **search**.

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. Agents that use **factored** or **structured** representations of states are called **planning agents**.

We distinguish between **informed** algorithms, in which the agent can estimate how far it is from the goal, and **uninformed** algorithms, where no such estimate is available.

Problem Statement:

- Imagine an agent in the city of Arad, Romania, enjoying a touring holiday.
- The agent's performance: it wants to improve its suntan, improve its Romanian, take in the sights, enjoy the nightlife, avoid hangovers, and so on.
- agent has a nonrefundable ticket to fly out of Bucharest the following day.
- Courses of action that don't reach Bucharest on time can be rejected without further consideration and the agent's decision problem is greatly simplified.
- The agent's task is to find out how to act, now and in the future, so that it reaches a goal state.
- Three roads lead out of Arad, one toward Sibiu, one to Timisoara, and one to Zerind.
- Agent should be familiar with the geography of Romania.
- Consider the agent has a map of Romania.
- map - provide the agent with information about the states it might get itself into and the actions it can take.
- environment is **observable (**agent always knows the current state)**,** environment is **discrete** (sign indicating its presence to arriving drivers)

Problem-Solving Agents:

If the agent has no additional information—that is, if the environment is **unknown**—then the agent can do no better than to execute one of the actions at random. For now, we assume that

our agents always have access to information about the world. With that information, the agent can follow this four-phase problem-solving process:

GOAL FORMULATION: Goals organize behaviour by limiting the objectives and hence the actions to be considered.

PROBLEM FORMULATION: The agent devises a description of the states and actions necessary to reach the goal—an abstract model of the relevant part of the world.

SEARCH: Before taking any action in the real world, the agent simulates sequences of actions in its model, searching until it finds a sequence of actions that reaches the goal. Such a sequence is called a **solution**.

EXECUTION: The agent can now execute the actions in the solution, one at a time.

It is an important property that in a fully observable, deterministic, known environment, the solution to any problem is a fixed sequence of actions. The **open-loop** system means that ignoring the precepts breaks the loop between agent and environment. If there is a chance that the model is incorrect, or the environment is nondeterministic, then the agent would be safer using a **closed-loop** approach that monitors the percepts.

In partially observable or nondeterministic environments, a solution would be a branching strategy that recommends different future actions depending on what percepts arrive.

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

Figure 3.1 A simple problem-solving agent. 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.

Well-defined problems and solutions(Search problems and solutions):

problem can be defined formally by five components
Initial state, actions, transaction model, goal test, path cost.

- **initial state:** agent starts in. initial state :Romania 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.
  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;
  function RESULT (s, a) that returns the state that results from doing action a in state s.
  successor refer to any state reachable from a given state by a single action.
  Example, RESULT (In (Arad),Go(Zerind)) = In(Zerind) .
  initial state, actions, and transition model implicitly define the state space of the problem
  State Space: 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 state space is a sequence of states connected by a sequence of actions.

- **Goal test**

  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. {In(Bucharest)}.

- **Path cost**

  Numeric cost to each path.
  sum of the costs of the individual actions along the path.
  The step cost c(s, a, s). assume that step costs are nonnegative.

Formulating problems:

The process of removing detail from a representation is called **abstraction**. The abstraction is valid if we can elaborate any abstract solution into a solution in the more detailed world. The abstraction is useful if carrying out each of the actions in the solution is easier than the original problem.

Example problem: Romania tour

- On holiday in Romania. Currently in Arad
- Flight leaves tomorrow at Bucharest
- Formulate the goal
  - Be in Bucharest
- Formulate the problem
  - State: various cities
  - Actions: Drive between cities
- Find Solution
  - Sequence of cities eg. Arad, Sibiu, Fagaras, Bucharest
- 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

- optimal solution has the lowest path cost among all solutions

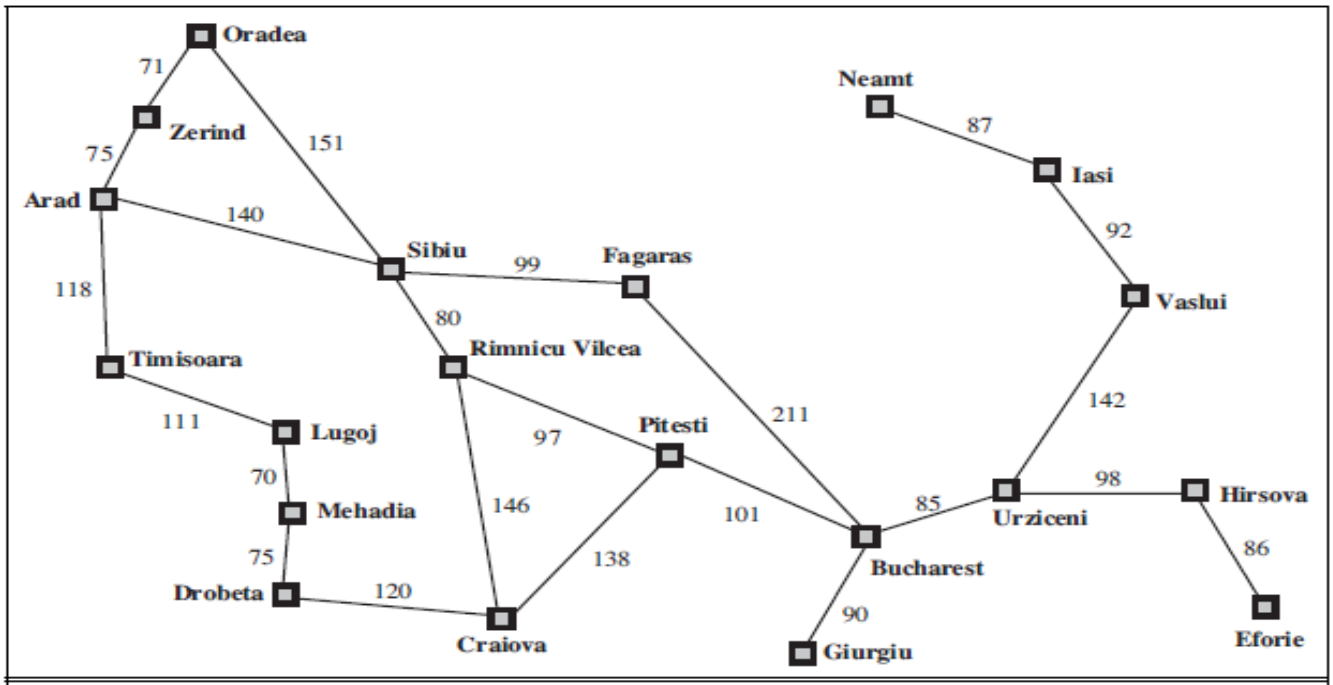


Figure 3.2 A simplified road map of part of Romania.

Single state problem formulation:

A problem is defined by 4 items

1.Intitial state: Eg" at Arad"
2.Actions or Sucessor function
s(x)=set of action state pairs
Eg: S(Arad)={<Arad->Zerind,Zerind>,….}
3.Goal test
Explicit eg x=at Burcharest"
Implicit eg:Chekmate(x)
4.Path cost function(additive)
Eg.Sum of distances,#actions executes,etc
C(x,a,y) is the step cost ,assumed to be >or equal to 0
5.A solution is a sequence of actions leading from the initial state to a goal state.

Selecting a State Space:

- Real world is absurdly complex; therefore state space must be abstracted for problem solving (Abstract) state=set of real states
- (Abstract) action=complex combination of real actions
- Ex:Arad->Zerind compelx set of possible routes, detours,rest stops etc.
- For guaranteed reliability 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 PROBLEMS:

A **standardized problem** is intended to illustrate or exercise various problem-solving methods. It can be given a concise, exact description and hence is suitable as a benchmark for researchers to compare the performance of algorithms. A **real-world problem**, such as robot navigation, is one whose solutions people actually use, and whose formulation is idiosyncratic, not standardized, because, for example, each robot has different sensors that produce different data.

- Toy problem :illustrate or exercise problem-solving methods
- A real-world problem is one whose solutions people actually care about.

Standardized problems:

A **grid world** problem is a two-dimensional rectangular array of square cells in which agents can move from cell to cell.

- **Vacuum world**
- **Sokoban puzzle**
- **Sliding-tile puzzle**

1.Toy Problem:

1. States: Dirt and robot location.
2. Actions: left, right, suck.
3. Goal test: no dirt at all locations.
4. Path cost :1 per action.

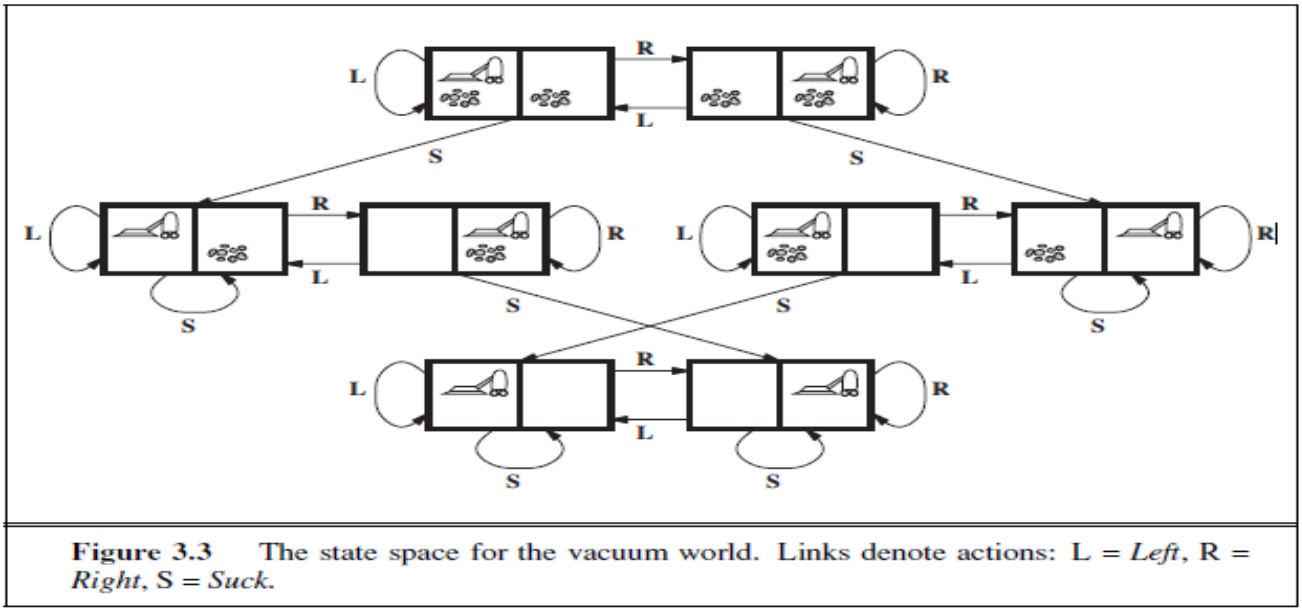


Figure 3.3 The state space for the vacuum world. Links denote actions: L = *Left*, R = *Right*, S = *Suck*.

1.States: Dirt and robot location
2.Initial state: Any state can be designated as the initial state.
3.Actions: In this simple environment, each state has just three actions: Left, Right, and Suck. Larger environments might also include Up and Down.
4.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.
5.Goal test: This checks whether all the squares are clean
6.Path cost: Each step costs 1, so the path cost is the number of steps in the path.

2.8-puzzle:

States: A state description specifies the location of each of the eight tiles and the blank in one of the nine squares.
Initial state: Any state can be designated as the initial state.
Actions: Left, Right, Up, or Down.
Transition model: Given a state and action, this returns the resulting state; for example, if we apply Left to the start state in Figure 3.4, the resulting state has the 5 and the blank switched.
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

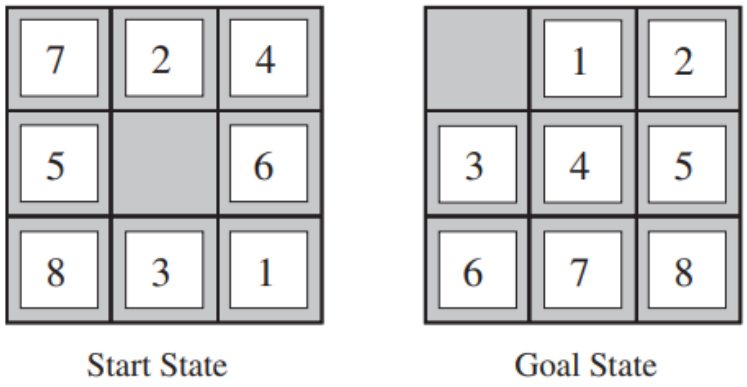


Figure 3.4 A typical instance of the 8-puzzle.

3.8-queens problem:

- 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.

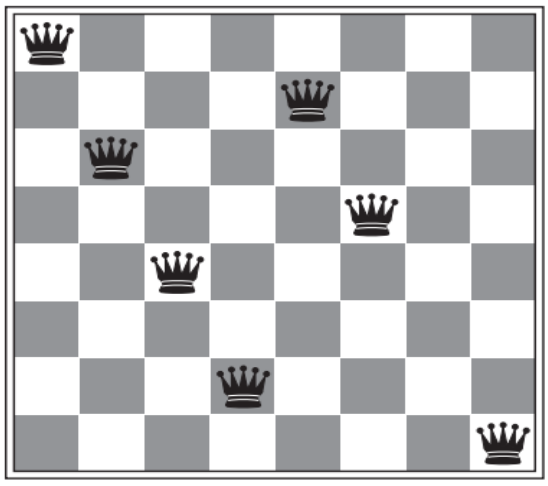


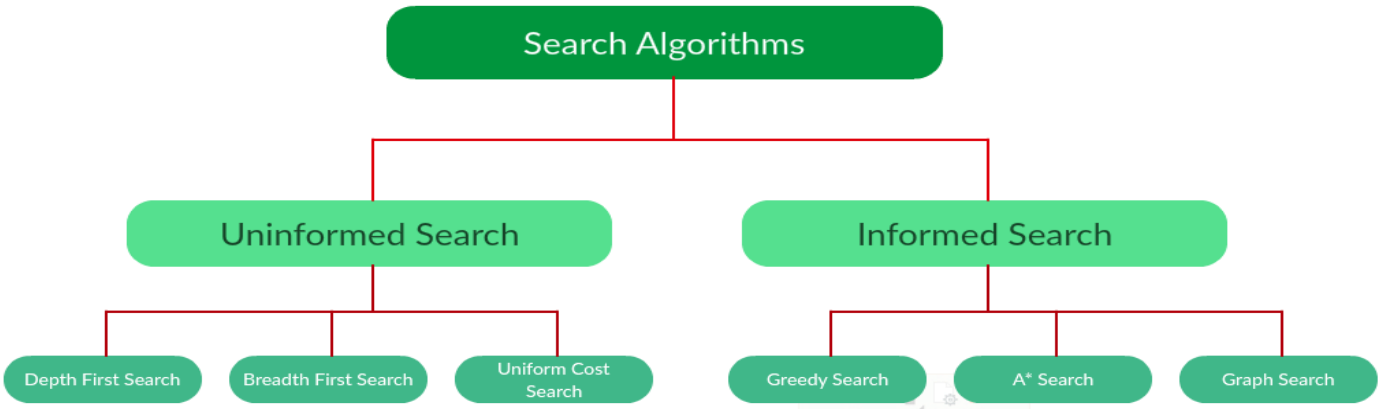
Figure 3.5 Almost a solution to the 8-queens problem. (Solution is left as an exercise.)

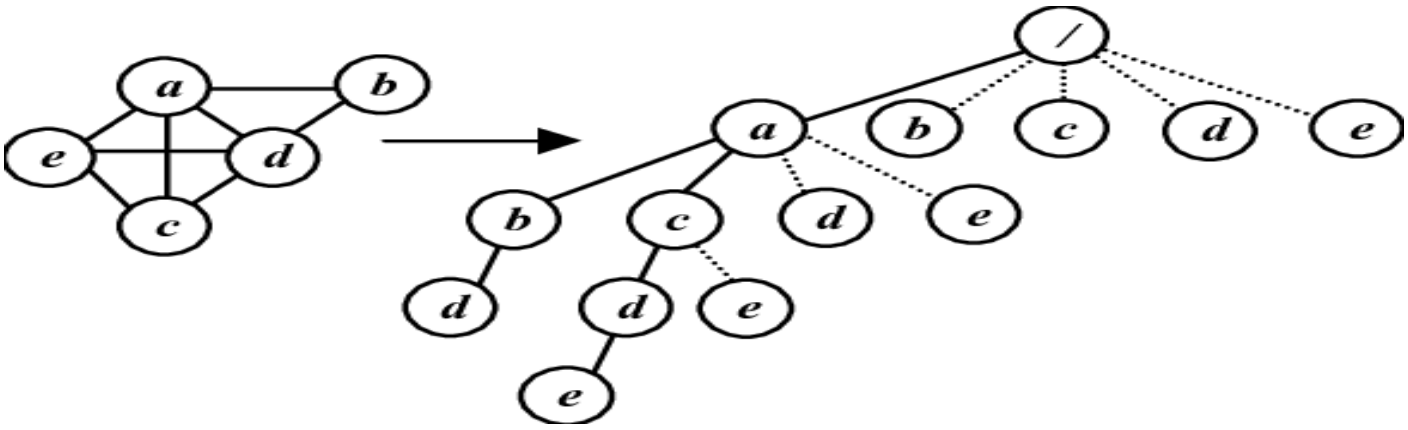
Real-world problems:

1.Route finding problems
GPS based navigation systems GMAP
2. Touring Problems
TSP
3.VLSI Layout Problem: millions of components and connections on a chip to minimize area, minimize circuit delays, minimize stray capacitances. cell layout and channel routing
4. Robot Navigation Problem
5.Automatic assembly sequencing:
6.Internet Searching
7.Searching paths in metabolic networks in bioinformatics: protein design, in which the goal is to find a sequence of amino acids that will fold into a three-dimensional protein with the right properties to cure some disease.

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.
- a route from Arad to Bucharest.
- The root node of the tree initial state, In(Arad).
- The first step is to test whether this is a goal state.
- Do this by expanding the current state; thereby generating a new set of states





- Add three branches from the parent node In(Arad) leading to three new child nodes: In(Sibiu), In(Timisoara), and In(Zerind).
- Suppose we choose Sibiu first.
- Whether it is a goal state (it is not) and then expand it to get In(Arad), In(Fagaras), In(Oradea), and In(RimnicuVilcea).
- Choose any of these four or go back and choose Timisoara or Zerind.
- Each of these six nodes is a leaf node, that is, 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.
- it includes the path from Arad to Sibiu and back to Arad again!
- *In (*Arad*)* is a **repeated state** in the search tree, generated in this case by a **loopy path**.
- Considering such loopy paths means that the complete search tree for Romania is *infinite* because there **is no limit** to how often one can traverse a loop.
- Loopy paths are a special case **redundant paths** whenever there is more than one way to get from one state to another.
- Consider the paths Arad–Sibiu (140 km long) and Arad–Zerind–Oradea–Sibiu (297 km long).
- second path is redundant

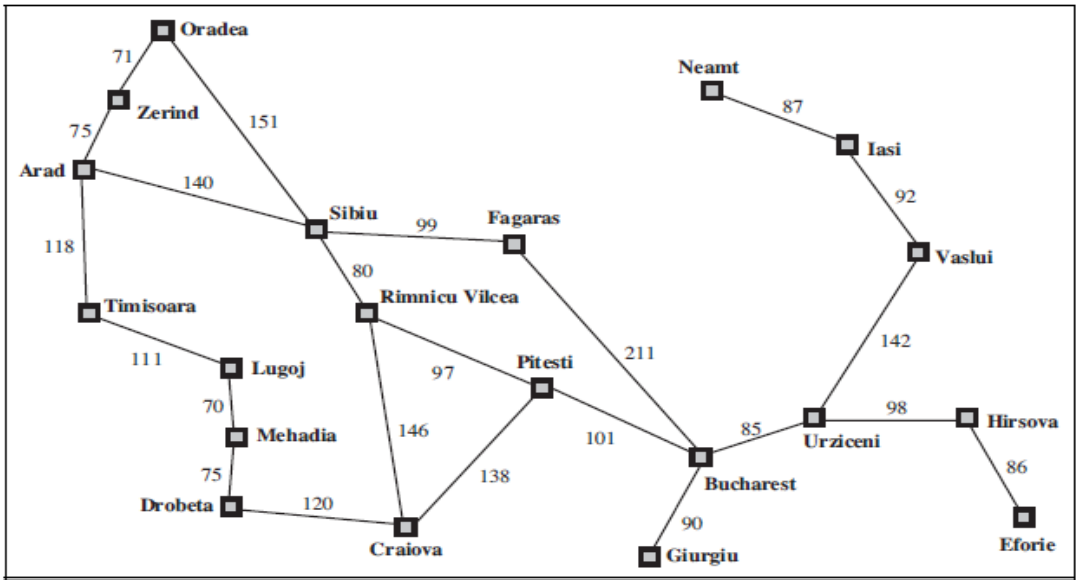


Figure 3.2 A simplified road map of part of Romania.

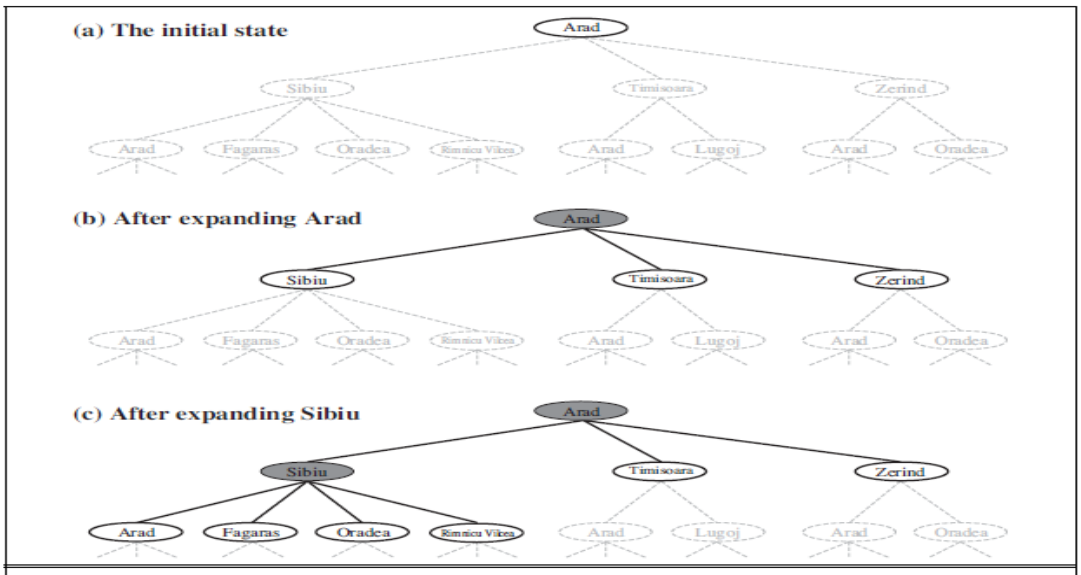


Figure 3.6 Partial search trees for finding a route from Arad to Bucharest. Nodes that have been expanded are shaded; nodes that have been generated but not yet expanded are outlined in bold; nodes that have not yet been generated are shown in faint dashed lines.