Contents

1 INTRODUCTION

In which we try to explain why we consider artificial intelligence to be a subject most worthy of study, and in which we try to decide what exactly it is, this being a good thing to decide before embarking.

We call ourselves *Homo sapiens*—man the wise—because our mental capacities are so important to us. For thousands of years, we have tried to understand *how we think*; that is, how a mere handful of stuff can perceive, understand, predict, and manipulate a world far larger and more complicated than itself. The field of **artificial intelligence**, or AI, goes further still: it attempts not just to understand but also to *build* intelligent entities.

ARTIFICIAL INTELLIGENCE

AI is one of the newest sciences. Work started in earnest soon after World War II, and the name itself was coined in 1956. Along with molecular biology, AI is regularly cited as the "field I would most like to be in" by scientists in other disciplines. A student in physics might reasonably feel that all the good ideas have already been taken by Galileo, Newton, Einstein, and the rest. AI, on the other hand, still has openings for several full-time Einsteins.

AI currently encompasses a huge variety of subfields, ranging from general-purpose areas, such as learning and perception to such specific tasks as playing chess, proving mathematical theorems, writing poetry, and diagnosing diseases. AI systematizes and automates intellectual tasks and is therefore potentially relevant to any sphere of human intellectual activity. In this sense, it is truly a universal field.

1.1 WHAT IS AI?

We have claimed that AI is exciting, but we have not said what it *is*. Definitions of artificial intelligence according to eight textbooks are shown in Figure 1.1. These definitions vary along two main dimensions. Roughly, the ones on top are concerned with *thought processes* and *reasoning*, whereas the ones on the bottom address *behavior*. The definitions on the left measure success in terms of fidelity to *human* performance, whereas the ones on the right measure against an *ideal* concept of intelligence, which we will call **rationality**. A system is rational if it does the "right thing," given what it knows.

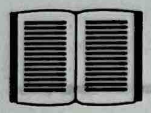
RATIONALITY

and background information.

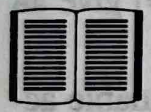
EXERCISES

These exercises are intended to stimulate discussion, and some might be set as term projects. Alternatively, preliminary attempts can be made now, and these attempts can be reviewed after the completion of the book.

1.1 Define in your own words: (a) intelligence, (b) artificial intelligence, (c) agent.



1.2 Read Turing's original paper on AI (Turing, 1950). In the paper, he discusses several potential objections to his proposed enterprise and his test for intelligence. Which objections still carry some weight? Are his refutations valid? Can you think of new objections arising from developments since he wrote the paper? In the paper, he predicts that, by the year 2000, a computer will have a 30% chance of passing a five-minute Turing Test with an unskilled interrogator. What chance do you think a computer would have today? In another 50 years?



1.3 Every year the Loebner prize is awarded to the program that comes closest to passing a version of the Turing test. Research and report on the latest winner of the Loebner prize. What techniques does it use? How does it advance the state of the art in AI?

1.4 There are well-known classes of problems that are intractably difficult for computers, and other classes that are provably undecidable. Does this mean that AI is impossible?

1.5 Suppose we extend Evans's ANALOGY program so that it can score 200 on a standard IQ test. Would we then have a program more intelligent than a human? Explain.

1.6 How could introspection—reporting on one's inner thoughts—be inaccurate? Could I be wrong about what I'm thinking? Discuss.

1.7 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.
c. Buying a week's worth of groceries at the market.
d. Buying a week's worth of groceries on the web.
e. Playing a decent game of bridge at a competitive level.
f. Discovering and proving new mathematical theorems.
g. Writing an intentionally funny story.
h. Giving competent legal advice in a specialized area of law.
i. Translating spoken English into spoken Swedish in real time.
j. Performing a complex surgical operation.

For the currently infeasible tasks, try to find out what the difficulties are and predict when, if ever, they will be overcome.

1.8 Some authors have claimed that perception and motor skills are the most important part of intelligence, and that "higher level" capacities are necessarily parasitic—simple add-ons to these underlying facilities. Certainly, most of evolution and a large part of the brain have been devoted to perception and motor skills, whereas AI has found tasks such as game playing and logical inference to be easier, in many ways, than perceiving and acting in the real world. Do you think that AI's traditional focus on higher-level cognitive abilities is misplaced?

1.9 Why would evolution tend to result in systems that act rationally? What goals are such systems designed to achieve?

1.10 Are reflex actions (such as moving your hand away from a hot stove) rational? Are they intelligent?

1.11 "Surely computers cannot be intelligent—they can do only what their programmers tell them." Is the latter statement true, and does it imply the former?

1.12 "Surely animals cannot be intelligent—they can do only what their genes tell them." Is the latter statement true, and does it imply the former?

1.13 "Surely animals, humans, and computers cannot be intelligent—they can do only what their constituent atoms are told to do by the laws of physics." Is the latter statement true, and does it imply the former?

2 INTELLIGENT AGENTS

In which we discuss the nature of agents, perfect or otherwise, the diversity of environments, and the resulting menagerie of agent types.

Chapter 1 identified the concept of **rational agents** as central to our approach to artificial intelligence. In this chapter, we make this notion more concrete. We will see that the concept of rationality can be applied to a wide variety of agents operating in any imaginable environment. Our plan in this book is to use this concept to develop a small set of design principles for building successful agents—systems that can reasonably be called **intelligent**.

We will begin by examining agents, environments, and the coupling between them. The observation that some agents behave better than others leads naturally to the idea of a rational agent—one that behaves as well as possible. How well an agent can behave depends on the nature of the environment; some environments are more difficult than others. We give a crude categorization of environments and show how properties of an environment influence the design of suitable agents for that environment. We describe a number of basic "skeleton" agent designs, which will be fleshed out in the rest of the book.

2.1 AGENTS AND ENVIRONMENTS

ENVIRONMENT
SENSOR
ACTUATOR

An **agent** is anything that can be viewed as perceiving its **environment** through **sensors** and acting upon that environment through **actuators**. This simple idea is illustrated in Figure 2.1. A human agent has eyes, ears, and other organs for sensors and hands, legs, mouth, and other body parts 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. We will make the general assumption that every agent can perceive its own actions (but not always the effects).

PERCEPT
PERCEPT SEQUENCE

We use the term **percept** to refer to the agent's perceptual inputs at any given instant. 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 the entire percept sequence observed to date.* If we can specify the agent's choice of action for every possible

The goal-based view also dominates the cognitive psychology tradition in the area of problem solving, beginning with the enormously influential *Human Problem Solving* (Newell and Simon, 1972) and running through all of Newell's later work (Newell, 1990). Goals, further analyzed as *desires* (general) and *intentions* (currently pursued), are central to the theory of agents developed by Bratman (1987). This theory has been influential both in natural language understanding and multiagent systems.

Horvitz *et al.* (1988) specifically suggest the use of rationality conceived as the maximization of expected utility as a basis for AI. The text by Pearl (1988) was the first in AI to cover probability and utility theory in depth; its exposition of practical methods for reasoning and decision making under uncertainty was probably the single biggest factor in the rapid shift towards utility-based agents in the 1990s (see Part V).

The general design for learning agents portrayed in Figure 2.15 is classic in the machine learning literature (Buchanan *et al.*, 1978; Mitchell, 1997). Examples of the design, as embodied in programs, go back at least as far as Arthur Samuel's (1959, 1967) learning program for playing checkers. Learning agents are discussed in depth in Part VI.

Interest in agents and in agent design has risen rapidly in recent years, partly because of the growth of the Internet and the perceived need for automated and mobile **softbots** (Etzioni and Weld, 1994). Relevant papers are collected in *Readings in Agents* (Huhns and Singh, 1998) and *Foundations of Rational Agency* (Wooldridge and Rao, 1999). *Multiagent Systems* (Weiss, 1999) provides a solid foundation for many aspects of agent design. Conferences devoted to agents include the International Conference on Autonomous Agents, the International Workshop on Agent Theories, Architectures, and Languages, and the International Conference on Multiagent Systems. Finally, *Dung Beetle Ecology* (Hanski and Cambefort, 1991) provides a wealth of interesting information on the behavior of dung beetles.

EXERCISES

2.1 Define in your own words the following terms: agent, agent function, agent program, rationality, autonomy, reflex agent, model-based agent, goal-based agent, utility-based agent, learning agent.

2.2 Both the performance measure and the utility function measure how well an agent is doing. Explain the difference between the two.

2.3 This exercise explores the differences between agent functions and agent programs.

a. Can there be more than one agent program that implements a given agent function? Give an example, or show why one is not possible.

b. Are there agent functions that cannot be implemented by any agent program?

c. Given a fixed machine architecture, does each agent program implement exactly one agent function?

d. Given an architecture with n bits of storage, how many different possible agent programs are there?

2.4 Let us examine the rationality of various vacuum-cleaner agent functions.

a. Show that the simple vacuum-cleaner agent function described in Figure 2.3 is indeed rational under the assumptions listed on page 36.

b. Describe a rational agent function for the modified performance measure that deducts one point for each movement. Does the corresponding agent program require internal state?

c. Discuss possible agent designs for the cases in which clean squares can become dirty and the geography of the environment is unknown. Does it make sense for the agent to learn from its experience in these cases? If so, what should it learn?

2.5 For each of the following agents, develop a PEAS description of the task environment:

a. Robot soccer player;

b. Internet book-shopping agent;

c. Autonomous Mars rover;

d. Mathematician's theorem-proving assistant.

2.6 For each of the agent types listed in Exercise 2.5, characterize the environment according to the properties given in Section 2.3, and select a suitable agent design.

The following exercises all concern the implementation of environments and agents for the vacuum-cleaner world.

2.7 Implement a performance-measuring environment simulator for the vacuum-cleaner world depicted in Figure 2.2 and specified on page 36. Your implementation should be modular, so that the sensors, actuators, and environment characteristics (size, shape, dirt placement, etc.) can be changed easily. (*Note:* for some choices of programming language and operating system there are already implementations in the online code repository.)

2.8 Implement a simple reflex agent for the vacuum environment in Exercise 2.7. Run the environment simulator with this agent for all possible initial dirt configurations and agent locations. Record the agent's performance score for each configuration and its overall average score.

2.9 Consider a modified version of the vacuum environment in Exercise 2.7, in which the agent is penalized one point for each movement.

a. Can a simple reflex agent be perfectly rational for this environment? Explain.

b. What about a reflex agent with state? Design such an agent.

c. How do your answers to **a** and **b** change if the agent's percepts give it the clean/dirty status of every square in the environment?

2.10 Consider a modified version of the vacuum environment in Exercise 2.7, in which the geography of the environment—its extent, boundaries, and obstacles—is unknown, as is the initial dirt configuration. (The agent can go *Up* and *Down* as well as *Left* and *Right*.)

a. Can a simple reflex agent be perfectly rational for this environment? Explain.

b. Can a simple reflex agent with a *randomized* agent function outperform a simple reflex agent? Design such an agent and measure its performance on several environments.

c. Can you design an environment in which your randomized agent will perform very poorly? Show your results.

d. Can a reflex agent with state outperform a simple reflex agent? Design such an agent and measure its performance on several environments. Can you design a rational agent of this type?

2.11 Repeat Exercise 2.10 for the case in which the location sensor is replaced with a "bump" sensor that detects the agent's attempts to move into an obstacle or to cross the boundaries of the environment. Suppose the bump sensor stops working; how should the agent behave?

2.12 The vacuum environments in the preceding exercises have all been deterministic. Discuss possible agent programs for each of the following stochastic versions:

a. Murphy's law: twenty-five percent of the time, the *Suck* action fails to clean the floor if it dirty and deposits dirt onto the floor if the floor is clean. How is your agent program affected if the dirt sensor gives the wrong answer 10% of the time?

b. Small children: At each time step, each clean square has a 10% chance of becoming dirty. Can you come up with a rational agent design for this case?

3 SOLVING PROBLEMS BY SEARCHING

In which we see how an agent can find a sequence of actions that achieves its goals, when no single action will do.

The simplest agents discussed in Chapter 2 were the reflex agents, which base their actions on a direct mapping from states to actions. Such agents cannot operate well in environments for which this mapping would be too large to store and would take too long to learn. Goal-based agents, on the other hand, can succeed by considering future actions and the desirability of their outcomes.

PROBLEM-SOLVING AGENT

This chapter describes one kind of goal-based agent called a **problem-solving agent**. Problem-solving agents decide what to do by finding sequences of actions that lead to desirable states. We start by defining precisely the elements that constitute a "problem" and its "solution," and give several examples to illustrate these definitions. We then describe several general-purpose search algorithms that can be used to solve these problems and compare the advantages of each algorithm. The algorithms are **uninformed**, in the sense that they are given no information about the problem other than its definition. Chapter 4 deals with **informed** search algorithms, ones that have some idea of where to look for solutions.

This chapter uses concepts from the analysis of algorithms. Readers unfamiliar with the concepts of asymptotic complexity (that is, $O()$ notation) and NP-completeness should consult Appendix A.

3.1 PROBLEM-SOLVING AGENTS

Intelligent agents are supposed to maximize their performance measure. As we mentioned in Chapter 2, achieving this is sometimes simplified if the agent can adopt a **goal** and aim at satisfying it. Let us first look at why and how an agent might do this.

Imagine an agent in the city of Arad, Romania, enjoying a touring holiday. The agent's performance measure contains many factors: it wants to improve its suntan, improve its 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 follow-

Uninformed search algorithms for problem solving are a central topic of classical computer science (Horowitz and Sahni, 1978) and operations research (Dreyfus, 1969); Deo and Pang (1984) and Gallo and Pallottino (1988) give more recent surveys. Breadth-first search was formulated for solving mazes by Moore (1959). The method of **dynamic programming** (Bellman and Dreyfus, 1962), which systematically records solutions for all subproblems of increasing lengths, can be seen as a form of breadth-first search on graphs. The two-point shortest-path algorithm of Dijkstra (1959) is the origin of uniform-cost search.

A version of iterative deepening designed to make efficient use of the chess clock was first used by Slate and Atkin (1977) in the CHESS 4.5 game-playing program, but the application to shortest path graph search is due to Korf (1985a). Bidirectional search, which was introduced by Pohl (1969, 1971), can also be very effective in some cases.

Partially observable and nondeterministic environments have not been studied in great depth within the problem-solving approach. Some efficiency issues in belief-state search have been investigated by Genesereth and Nourbakhsh (1993). Koenig and Simmons (1998) studied robot navigation from an unknown initial position, and Erdmann and Mason (1988) studied the problem of robotic manipulation without sensors, using a continuous form of belief-state search. Contingency search has been studied within the planning subfield. (See Chapter 12.) For the most part, planning and acting with uncertain information have been handled using the tools of probability and decision theory (see Chapter 17).

The textbooks by Nilsson (1971, 1980) are good general sources of information about classical search algorithms. A comprehensive and more up-to-date survey can be found in Korf (1988). Papers about new search algorithms—which, remarkably, continue to be discovered—appear in journals such as *Artificial Intelligence*.

EXERCISES

3.1 Define in your own words the following terms: state, state space, search tree, search node, goal, action, successor function, and branching factor.

3.2 Explain why problem formulation must follow goal formulation.

3.3 Suppose that LEGAL-ACTIONS(s) denotes the set of actions that are legal in state s, and RESULT(a, s) denotes the state that results from performing a legal action a in state s. Define SUCCESSOR-FN in terms of LEGAL-ACTIONS and RESULT, and *vice versa*.

3.4 Show that the 8-puzzle states are divided into two disjoint sets, such that no state in one set can be transformed into a state in the other set by any number of moves. (*Hint:* See Berlekamp *et al.* (1982).) Devise a procedure that will tell you which class a given state is in, and explain why this is a good thing to have for generating random states.

3.5 Consider the n-queens problem using the "efficient" incremental formulation given on page 67. Explain why the state space size is at least $\sqrt[3]{n!}$ and estimate the largest n for which exhaustive exploration is feasible. (*Hint*: Derive a lower bound on the branching factor by considering the maximum number of squares that a queen can attack in any column.)

3.6 Does a finite state space always lead to a finite search tree? How about a finite state space that is a tree? Can you be more precise about what types of state spaces always lead to finite search trees? (Adapted from Bender, 1996.)

3.7 Give the initial state, goal test, successor function, and cost function for each of the following. Choose a formulation that is precise enough to be implemented.

a. You have to color a planar map using only four colors, in such a way that no two adjacent regions have the same color.

b. A 3-foot-tall monkey is in a room where some bananas are suspended from the 8-foot ceiling. He would like to get the bananas. The room contains two stackable, movable, climbable 3-foot-high crates.

c. You have a program that outputs the message "illegal input record" when fed a certain file of input records. You know that processing of each record is independent of the other records. You want to discover what record is illegal.

d. You have three jugs, measuring 12 gallons, 8 gallons, and 3 gallons, and a water faucet. You can fill the jugs up or empty them out from one to another or onto the ground. You need to measure out exactly one gallon.

3.8 Consider a state space where the start state is number 1 and the successor function for state n returns two states, numbers $2n$ and $2n + 1$.

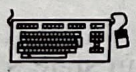
a. Draw the portion of the state space for states 1 to 15.

b. Suppose the goal state is 11. List the order in which nodes will be visited for breadth-first search, depth-limited search with limit 3, and iterative deepening search.

c. Would bidirectional search be appropriate for this problem? If so, describe in detail how it would work.

d. What is the branching factor in each direction of the bidirectional search?

e. Does the answer to (c) suggest a reformulation of the problem that would allow you to solve the problem of getting from state 1 to a given goal state with almost no search?



3.9 The **missionaries and cannibals** problem is usually stated as follows. Three missionaries and three cannibals are on one side of a river, along with a boat that can hold one or two people. Find a way to get everyone to the other side, without ever leaving a group of missionaries in one place outnumbered by the cannibals in that place. This problem is famous in AI because it was the subject of the first paper that approached problem formulation from an analytical viewpoint (Amarel, 1968).

a. Formulate the problem precisely, making only those distinctions necessary to ensure a valid solution. Draw a diagram of the complete state space.

b. Implement and solve the problem optimally using an appropriate search algorithm. Is it a good idea to check for repeated states?

c. Why do you think people have a hard time solving this puzzle, given that the state space is so simple?

3.10 Implement two versions of the successor function for the 8-puzzle: one that generates all the successors at once by copying and editing the 8-puzzle data structure, and one that generates one new successor each time it is called and works by modifying the parent state directly (and undoing the modifications as needed). Write versions of iterative deepening depth-first search that use these functions and compare their performance.

3.11 On page 79, we mentioned **iterative lengthening search**, an iterative analog of uniform cost search. The idea is to use increasing limits on path cost. If a node is generated whose path cost exceeds the current limit, it is immediately discarded. For each new iteration, the limit is set to the lowest path cost of any node discarded in the previous iteration.

a. Show that this algorithm is optimal for general path costs.

b. Consider a uniform tree with branching factor b, solution depth d, and unit step costs. How many iterations will iterative lengthening require?

c. Now consider step costs drawn from the continuous range $[0, 1]$ with a minimum positive cost ϵ. How many iterations are required in the worst case?

d. Implement the algorithm and apply it to instances of the 8-puzzle and traveling salesperson problems. Compare the algorithm's performance to that of uniform-cost search, and comment on your results.

3.12 Prove that uniform-cost search and breadth-first search with constant step costs are optimal when used with the GRAPH-SEARCH algorithm. Show a state space with constant step costs in which GRAPH-SEARCH using iterative deepening finds a suboptimal solution.

3.13 Describe a state space in which iterative deepening search performs much worse than depth-first search (for example, $O(n^2)$ vs. $O(n)$).

3.14 Write a program that will take as input two Web page URLs and find a path of links from one to the other. What is an appropriate search strategy? Is bidirectional search a good idea? Could a search engine be used to implement a predecessor function?

3.15 Consider the problem of finding the shortest path between two points on a plane that has convex polygonal obstacles as shown in Figure 3.22. This is an idealization of the problem that a robot has to solve to navigate its way around a crowded environment.

a. Suppose the state space consists of all positions (x, y) in the plane. How many states are there? How many paths are there to the goal?

b. Explain briefly why the shortest path from one polygon vertex to any other in the scene must consist of straight-line segments joining some of the vertices of the polygons. Define a good state space now. How large is this state space?

c. Define the necessary functions to implement the search problem, including a successor function that takes a vertex as input and returns the set of vertices that can be reached in a straight line from the given vertex. (Do not forget the neighbors on the same polygon.) Use the straight-line distance for the heuristic function.

d. Apply one or more of the algorithms in this chapter to solve a range of problems in the domain, and comment on their performance.

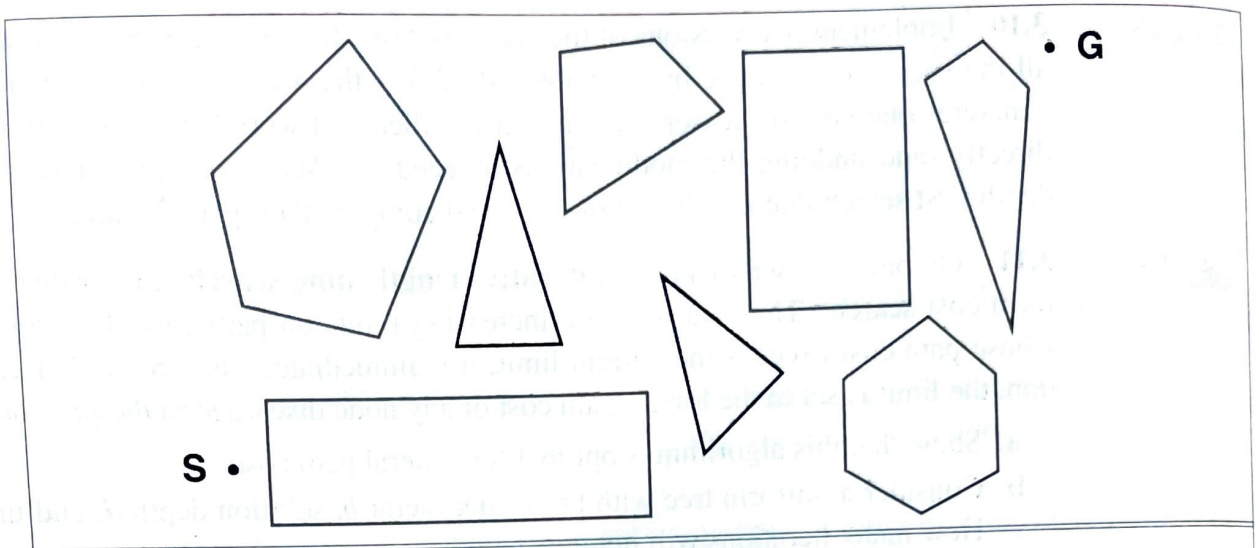


Figure 3.22 A scene with polygonal obstacles.

3.16 We can turn the navigation problem in Exercise 3.15 into an environment as follows:

- The percept will be a list of the positions, *relative to the agent*, of the visible vertices. The percept does *not* include the position of the robot! The robot must learn its own position from the map; for now, you can assume that each location has a different "view."
- Each action will be a vector describing a straight-line path to follow. If the path is unobstructed, the action succeeds; otherwise, the robot stops at the point where its path first intersects an obstacle. If the agent returns a zero motion vector and is at the goal (which is fixed and known), then the environment should teleport the agent to a *random location* (not inside an obstacle).
- The performance measure charges the agent 1 point for each unit of distance traversed and awards 1000 points each time the goal is reached.

a. Implement this environment and a problem-solving agent for it. The agent will need to formulate a new problem after each teleportation, which will involve discovering its current location.

b. Document your agent's performance (by having the agent generate suitable commentary as it moves around) and report its performance over 100 episodes.

c. Modify the environment so that 30% of the time the agent ends up at an unintended destination (chosen randomly from the other visible vertices if any, otherwise no move at all). This is a crude model of the motion errors of a real robot. Modify the agent so that when such an error is detected, it finds out where it is and then constructs a plan to get back to where it was and resume the old plan. Remember that sometimes getting back to where it was might also fail! Show an example of the agent successfully overcoming two successive motion errors and still reaching the goal.

d. Now try two different recovery schemes after an error: (1) Head for the closest vertex on the original route; and (2) replan a route to the goal from the new location. Compare the performance of the three recovery schemes. Would the inclusion of search costs affect the comparison?

e. Now suppose that there are locations from which the view is identical. (For example, suppose the world is a grid with square obstacles.) What kind of problem does the agent now face? What do solutions look like?

3.17 On page 62, we said that we would not consider problems with negative path costs. In this exercise, we explore this in more depth.

a. Suppose that actions can have arbitrarily large negative costs; explain why this possibility would force any optimal algorithm to explore the entire state space.

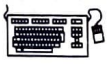
b. Does it help if we insist that step costs must be greater than or equal to some negative constant c? Consider both trees and graphs.

c. Suppose that there is a set of operators that form a loop, so that executing the set in some order results in no net change to the state. If all of these operators have negative cost, what does this imply about the optimal behavior for an agent in such an environment?

d. One can easily imagine operators with high negative cost, even in domains such as route finding. For example, some stretches of road might have such beautiful scenery as to far outweigh the normal costs in terms of time and fuel. Explain, in precise terms, within the context of state-space search, why humans do not drive round scenic loops indefinitely, and explain how to define the state space and operators for route finding so that artificial agents can also avoid looping.

e. Can you think of a real domain in which step costs are such as to cause looping?

3.18 Consider the sensorless, two-location vacuum world under Murphy's Law. Draw the belief state space reachable from the initial belief state $\{1, 2, 3, 4, 5, 6, 7, 8\}$, and explain why the problem is unsolvable. Show also that if the world is fully observable then there is a solution sequence for each possible initial state.



3.19 Consider the vacuum-world problem defined in Figure 2.2.

a. Which of the algorithms defined in this chapter would be appropriate for this problem? Should the algorithm check for repeated states?

b. Apply your chosen algorithm to compute an optimal sequence of actions for a 3×3 world whose initial state has dirt in the three top squares and the agent in the center.

c. Construct a search agent for the vacuum world, and evaluate its performance in a set of 3×3 worlds with probability 0.2 of dirt in each square. Include the search cost as well as path cost in the performance measure, using a reasonable exchange rate.

d. Compare your best search agent with a simple randomized reflex agent that sucks if there is dirt and otherwise moves randomly.

e. Consider what would happen if the world were enlarged to $n \times n$. How does the performance of the search agent and of the reflex agent vary with n?

4 INFORMED SEARCH AND EXPLORATION

In which we see how information about the state space can prevent algorithms from blundering about in the dark.

Chapter 3 showed that uninformed search strategies can find solutions to problems by systematically generating new states and testing them against the goal. Unfortunately, these strategies are incredibly inefficient in most cases. This chapter shows how an informed search strategy—one that uses problem-specific knowledge—can find solutions more efficiently. Section 4.1 describes informed versions of the algorithms in Chapter 3, and Section 4.2 explains how the necessary problem-specific information can be obtained. Sections 4.3 and 4.4 cover algorithms that perform purely **local search** in the state space, evaluating and modifying one or more current states rather than systematically exploring paths from an initial state. These algorithms are suitable for problems in which the path cost is irrelevant and all that matters is the solution state itself. The family of local-search algorithms includes methods inspired by statistical physics (**simulated annealing**) and evolutionary biology (**genetic algorithms**). Finally, Section 4.5 investigates **online search**, in which the agent is faced with a state space that is completely unknown.

4.1 INFORMED (HEURISTIC) SEARCH STRATEGIES

INFORMED SEARCH

This section shows how an **informed search** strategy—one that uses problem-specific knowledge beyond the definition of the problem itself—can find solutions more efficiently than an uninformed strategy.

BEST-FIRST SEARCH

The general approach we will consider is called **best-first search**. Best-first search is an instance of the general TREE-SEARCH or GRAPH-SEARCH algorithm in which a node is selected for expansion based on an **evaluation function**, $f(n)$. Traditionally, the node with the *lowest* evaluation is selected for expansion, because the evaluation measures distance to the goal. Best-first search can be implemented within our general search framework via a priority queue, a data structure that will maintain the fringe in ascending order of f-values.

EVALUATION FUNCTION

The name "best-first search" is a venerable but inaccurate one. After all, if we could *really* expand the best node first, it would not be a search at all; it would be a straight march to

a small competitive ratio is achievable with square obstacles, but with general rectangular obstacles no bounded ratio can be achieved. (See Figure 4.19.)

REAL-TIME SEARCH

The LRTA* algorithm was developed by Korf (1990) as part of an investigation into **real-time search** for environments in which the agent must act after searching for only a fixed amount of time (a much more common situation in two-player games). LRTA* is in fact a special case of reinforcement learning algorithms for stochastic environments (Barto *et al.*, 1995). Its policy of optimism under uncertainty—always head for the closest unvisited state—can result in an exploration pattern that is less efficient in the uninformed case than simple depth-first search (Koenig, 2000). Dasgupta *et al.* (1994) show that online iterative deepening search is optimally efficient for finding a goal in a uniform tree with no heuristic information. Several informed variants on the LRTA* theme have been developed with different methods for searching and updating within the known portion of the graph (Pemberton and Korf, 1992). As yet, there is no good understanding of how to find goals with optimal efficiency when using heuristic information.

PARALLEL SEARCH

The topic of **parallel search** algorithms was not covered in the chapter, partly because it requires a lengthy discussion of parallel computer architectures. Parallel search is becoming an important topic in both AI and theoretical computer science. A brief introduction to the AI literature can be found in Mahanti and Daniels (1993).

EXERCISES

4.1 Trace the operation of A* search applied to the problem of getting to Bucharest from Lugoj using the straight-line distance heuristic. That is, show the sequence of nodes that the algorithm will consider and the f, g, and h score for each node.

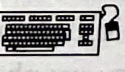
4.2 The **heuristic path algorithm** is a best-first search in which the objective function is $f(n) = (2 - w)g(n) + wh(n)$. For what values of w is this algorithm guaranteed to be optimal? What kind of search does this perform when $w = 0$? When $w = 1$? When $w = 2$?

4.3 Prove each of the following statements:

a. Breadth-first search is a special case of uniform-cost search.

b. Breadth-first search, depth-first search, and uniform-cost search are special cases of best-first search.

c. Uniform-cost search is a special case of A* search.



4.4 Devise a state space in which A* using GRAPH-SEARCH returns a suboptimal solution with an $h(n)$ function that is admissible but inconsistent.

4.5 We saw on page 96 that the straight-line distance heuristic leads greedy best-first search astray on the problem of going from Iasi to Fagaras. However, the heuristic is perfect on the opposite problem: going from Fagaras to Iasi. Are there problems for which the heuristic is misleading in both directions?

4.6 Invent a heuristic function for the 8-puzzle that sometimes overestimates, and show how it can lead to a suboptimal solution on a particular problem. (You can use a computer to help if you want.) Prove that, if h never overestimates by more than c, A* using h returns a solution whose cost exceeds that of the optimal solution by no more than c.

4.7 Prove that if a heuristic is consistent, it must be admissible. Construct an admissible heuristic that is not consistent.

4.8 The traveling salesperson problem (TSP) can be solved via the minimum spanning tree (MST) heuristic, which is used to estimate the cost of completing a tour, given that a partial tour has already been constructed. The MST cost of a set of cities is the smallest sum of the link costs of any tree that connects all the cities.

a. Show how this heuristic can be derived from a relaxed version of the TSP.

b. Show that the MST heuristic dominates straight-line distance.

c. Write a problem generator for instances of the TSP where cities are represented by random points in the unit square.

d. Find an efficient algorithm in the literature for constructing the MST, and use it with an admissible search algorithm to solve instances of the TSP.

4.9 On page 108, we defined the relaxation of the 8-puzzle in which a tile can move from square A to square B if B is blank. The exact solution of this problem defines **Gaschnig's heuristic** (Gaschnig, 1979). Explain why Gaschnig's heuristic is at least as accurate as h_1 (misplaced tiles), and show cases where it is more accurate than both h_1 and h_2 (Manhattan distance). Can you suggest a way to calculate Gaschnig's heuristic efficiently?

4.10 We gave two simple heuristics for the 8-puzzle: Manhattan distance and misplaced tiles. Several heuristics in the literature purport to improve on this—see, for example, Nilsson (1971), Mostow and Prieditis (1989), and Hansson *et al.* (1992). Test these claims by implementing the heuristics and comparing the performance of the resulting algorithms.

4.11 Give the name of the algorithm that results from each of the following special cases:

a. Local beam search with $k = 1$.

b. Local beam search with $k = \infty$.

c. Simulated annealing with $T = 0$ at all times.

d. Genetic algorithm with population size $N = 1$.

4.12 Sometimes there is no good evaluation function for a problem, but there is a good comparison method: a way to tell whether one node is better than another, without assigning numerical values to either. Show that this is enough to do a best-first search. Is there an analog of A*?

4.13 Relate the time complexity of LRTA* to its space complexity.

4.14 Suppose that an agent is in a 3×3 maze environment like the one shown in Figure 4.18. The agent knows that its initial location is (1,1), that the goal is at (3,3), and that the

four actions *Up*, *Down*, *Left*, *Right* have their usual effects unless blocked by a wall. The agent does *not* know where the internal walls are. In any given state, the agent perceives the set of legal actions; it can also tell whether the state is one it has visited before or a new state.

a. Explain how this online search problem can be viewed as an offline search in belief state space, where the initial belief state includes all possible environment configurations. How large is the initial belief state? How large is the space of belief states?

b. How many distinct percepts are possible in the initial state?

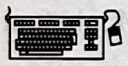
c. Describe the first few branches of a contingency plan for this problem. How large (roughly) is the complete plan?

Notice that this contingency plan is a solution for *every possible environment* fitting the given description. Therefore, interleaving of search and execution is not strictly necessary even in unknown environments.

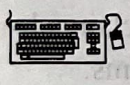
4.15 In this exercise, we will explore the use of local search methods to solve TSPs of the type defined in Exercise 4.8.

a. Devise a hill-climbing approach to solve TSPs. Compare the results with optimal solutions obtained via the A* algorithm with the MST heuristic (Exercise 4.8).

b. Devise a genetic algorithm approach to the traveling salesperson problem. Compare results to the other approaches. You may want to consult Larrañaga *et al.* (1999) for some suggestions for representations.



4.16 Generate a large number of 8-puzzle and 8-queens instances and solve them (where possible) by hill climbing (steepest-ascent and first-choice variants), hill climbing with random restart, and simulated annealing. Measure the search cost and percentage of solved problems and graph these against the optimal solution cost. Comment on your results.



4.17 In this exercise, we will examine hill climbing in the context of robot navigation, using the environment in Figure 3.22 as an example.

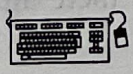
a. Repeat Exercise 3.16 using hill climbing. Does your agent ever get stuck in a local minimum? Is it *possible* for it to get stuck with convex obstacles?

b. Construct a nonconvex polygonal environment in which the agent gets stuck.

c. Modify the hill-climbing algorithm so that, instead of doing a depth-1 search to decide where to go next, it does a depth-k search. It should find the best k-step path and do one step along it, and then repeat the process.

d. Is there some k for which the new algorithm is guaranteed to escape from local minima?

e. Explain how LRTA* enables the agent to escape from local minima in this case.



4.18 Compare the performance of A* and RBFS on a set of randomly generated problems in the 8-puzzle (with Manhattan distance) and TSP (with MST—see Exercise 4.8) domains. Discuss your results. What happens to the performance of RBFS when a small random number is added to the heuristic values in the 8-puzzle domain?