UNIVERSITY
OF LONDON

Artificial intelligence

G. Konidaris

CO3310

2013

Undergraduate study in
Computing and related programmes

Goldsmiths
UNIVERSITY OF LONDON

This guide was prepared for the University of London by:

George Konidaris

The guide was produced by Sarah Rauchas, Department of Computing, Goldsmiths, University of London

This is one of a series of subject guides published by the University. We regret that due to pressure of work the author is unable to enter into any correspondence relating to, or arising from, the guide. If you have any comments on this subject guide, favourable or unfavourable, please use the form at the back of this guide.

In this and other publications you may see references to the 'University of London International Programmes', which was the name of the University's flexible and distance learning arm until 2018. It is now known simply as the 'University of London', which better reflects the academic award our students are working towards. The change in name will be incorporated into our materials as they are revised.

University of London
Publications Office
32 Russell Square
London WC1B 5DN
United Kingdom

london.ac.uk

Published by: University of London

Contents

Chapter 1

Introduction

Essential reading

Russell, S. and P. Norvig *Artificial Intelligence: A Modern Approach*. (Upper Saddle River, NJ: Prentice Hall, c2010) third edition [ISBN 9780132071482 (pbk); 9780136042594 (hbk)]. Chapter 1. This very readable introduction covers common definitions of AI and its goals, foundational developments in other fields that led to the emergence of AI, and its history as a distinct field.

1.1 Unit introduction

Welcome to this unit—**Artificial Intelligence**. Through this unit we hope to introduce you to an exciting, dynamic, and relatively young field that deals with one of the most important open questions in science, and has the potential to make a huge impact on our daily lives.

Despite its relative youth, Artificial Intelligence (or **AI**) is a large field with several active subfields, each with their own textbooks, journals and conference. Our aim in this unit is to provide you with a broad foundation in, and good general knowledge of, the basics of the field. We hope to provide a sufficient basis in the field to allow you to understand how to apply AI approaches to real-life problems that you may encounter in your future career. We also hope to shed some light on how we might go about building an intelligent system, and through this provide some insight into how humans think.

1.2 What is Artificial intelligence?

First, we should start by trying to define AI. Broadly speaking, AI is the study and design of computer systems that exhibit intelligence.

However, in practice there is a great deal of variation in what is considered intelligent, and how intelligence should be achieved. Russell and present eight different definitions of the field that reflect these differences:

1. ' ... effort to make computers think ... ' (Haugeland, 1985)
2. '[The automation of] activities that we associate with human thinking ...' (Bellman, 1978)
3. 'The study of mental faculties through the use of computer models.' (Charniak and McDermott, 1985)

4. 'The study of the computations that make it possible to perceive, reason, and act.' (Winston, 1992)

5. ' ... creating machines that perform functions that require intelligence when performed by people.' (Kurzweil, 1990)

6. ' ... how to make computers do things at which, at the moment, people are better.' (Rich and Knight, 1991)

7. ' ... the study of the design of intelligence agents.' (Poole et al., 1998)

8. ' ... intelligence behaviour in artifacts.' (Nilsson, 1998)

The notion that underlies all of these approaches is that **the brain is a computer**: its functions can be understood as computational processes, and can therefore be formulated and synthesised as computer programs. Therefore, a great deal of the material you will deal with in this unit aims to formalise our notions of how we think, and use that formalisation to develop working computer systems aimed at replicating our abilities.

Learning activity

Which of the above definitions most closely matches your intuitive idea of what AI is? Write a short paragraph explaining why, pointing out deficiencies in the definition, and briefly outlining one or two possible criticisms of it.

1.3 What is the goal of AI?

One important early goal of AI was to understand and replicate human thought processes through computational modelling. However, research in this area is now mostly considered to fall under the discipline of **Cognitive Science**.

Another goal, typified by Alan Turing's famous 'Turing test' for intelligence—in which a program is judged intelligent if its behaviour cannot be differentiated from that of a human (and which will be further discussed in Chapter 8)—is to explicitly aim to replicate human behaviour, albeit without regard to mechanism.

However, most modern AI programs are instead designed to **act rationally**, meaning that they take the **best possible action** given their goals, knowledge and constraints. This is an important goal formulation for three reasons.

First, it provides a concrete, objective measure against which performance can be judged and understood; second, it releases the field from having to model human idiosyncrasies, some of which are poorly understood; and finally, it leads to systems that may be useful in situations in which humans do **not** always behave rationally.

One of the skills you will learn through this subject is the ability to precisely formulate a problem and identify quantitative measures of its solution quality that can be used to effectively solve it.

1.4 Subfields

Most work in AI focuses on smaller components thought to be necessary for producing intelligent programs. The major subfields, some of which will be considered in the following chapters, are:

1. **Problem solving**, where an agent is given a problem setting and a goal and must determine how to realize that goal. For example, given the axioms of number theory, an agent could try to prove that there are infinitely many prime numbers.
2. **Knowledge representation and reasoning**, which studies how an agent can represent knowledge it has about the environment and use it to derive further knowledge, either using a logic-based representation (when the knowledge is certain) or a probabilistic one (when it is uncertain).
3. **Planning**, where an agent is given knowledge of an environment and must formulate a plan for interacting with it to achieve its goals.
4. **Learning**, where an agent must improve its performance through experience. This can take the form of learning to distinguish between categories of objects (supervised learning), learning structure from raw data (unsupervised learning), or learning to maximise reward (or minimise cost) in an environment with which the agent can interact (reinforcement learning).
5. **Vision**, where an agent must interpret or otherwise process raw visual images.
6. **Natural language**, where an agent must process input in a natural language (e.g. English), or generate it.

The technology resulting from progress in these areas can also typically be applied to otherwise conventional problems that cannot be solved using standard methods—problems in which a solution requires some aspect of what we might consider intelligence.

Solving such 'hard problems' provides both practical applications for AI research and test beds on which newly developed methods may be validated. Thus, much of AI is concerned with solving such problems in specialised domains (for example, using planning to schedule a very efficient work flow in a production line) without building a system that demonstrates general intelligence.

Learning activity

Pick one of the subfields above, and consider how the activity it represents happens in your brain. Write down a few day-to-day examples where you perform the activity, and try to list all of the information about the world that you need to know to do so successfully.

1.5 Reading advice and other resources

Reading for this unit is always split into Essential reading and Further reading. Essential reading forms the core of this subject, so you should read all chapters

indicated as Essential reading and ensure that you understand them. All of the Essential reading for this subject is from a single book:

- Russell, S. and P. Norvig *Artificial Intelligence: A Modern Approach*. (Upper Saddle River, NJ: Prentice Hall, c2010) third edition [ISBN 9780132071482 (pbk); 9780136042594 (hbk)].

Russell and Norvig is one of the standard AI textbooks and covers a great deal of material; although you may enjoy reading all of it, you do not need to. The chapters that you should read are identified in the Essential reading list at the beginning of each chapter of this subject guide.

Further reading items are also included for each chapter of this guide. You should use these sources, if they are available, to broaden your knowledge of the specific topic of the chapter and to augment the material available in the subject guide and Essential reading.

You should also feel free to make use of material on the Internet to aid you in your understanding of this subject. In particular, Wikipedia (www.wikipedia.org) and Scholarpedia (www.scholarpedia.org) often have helpful articles on AI subjects.

Most chapters include exercises designed to deepen your understanding of the material presented in that chapter. You should attempt to complete every exercise, to the best of your ability. This guide also includes a Sample examination paper and example solution (in the Appendix) that you can use as preparation for your final examination.

Please refer to the Computing VLE for other resources, including past examination papers and **Examiner's reports** for this subject, which should be used as an aid to your learning.

1.6 About this guide

This subject guide is not a subject text. It sets out a sequence of study for the topics in the subject, and provides a high-level overview of them. It also provides guidance for further reading. It is **not** a definitive statement of the material in the subject, nor does it cover the material at the same level of depth as the unit.

Students should be prepared to be examined on any topic which can reasonably be seen to lie within the remit of the syllabus, whether or not it is specifically mentioned in the subject guide.

Chapter 2

Intelligent agents

Essential reading

Russell, S. and P. Norvig *Artificial Intelligence: A Modern Approach*. (Upper Saddle River, NJ: Prentice Hall, c2010) third edition [ISBN 9780132071482 (pbk); 9780136042594 (hbk)]. Chapter 2.

2.1 Introduction

As we saw in the last chapter, AI is principally concerned with constructing systems that behave rationally. We can model such a system as an **agent**.

Agents are systems that interact with an **environment** using **sensors** to receive perceptual inputs (called **percepts**) from it, and **actuators** to act upon it.

A percept generally describes the state of an agent's sensors at a given moment in time. The sequence of all percepts received by the agent is called the agent's **perceptual sequence**.

2.1.1 The agent program

In AI, our goal is the construction of agents that behave rationally. This requires us to specify an agent's behaviour. We accomplish this by building an **agent program** that performs decision making and action selection.

Mathematically, an agent program maps an agent's perceptual sequence to an action. Such a mapping could, in principle, be described by an **agent function table** that explicitly lists an action for every possible combination of perceptual history.

However, such a table is in most cases either infeasibly large or even impossible to actually construct. In practice, the agent program is just that—a program.

2.1.2 Examples

Some examples of agents are:

- **A helicopter control agent** has altitude, speed and pose readings as percepts, rotor speeds as actuators, and has the sky as an environment. The agent program is a control program that controls the rotors to manouevre the helicopter to its goal.

- **A line assembly robot** has position information as percepts, arm and gripper motors as actuators, and a factory floor as an environment. The agent program sequences the robot's movements to assemble a product from a set of component parts.

- **A web search agent** has English search queries as percepts, accesses to a database of web pages and an output web page as actuators, and the Internet as an environment. The agent program queries the database to find the best set of matches to the search queries, and displays them on the output web page.

Learning activity

Consider a mouse. What are its sensors, actuators, and environment? Discuss how its sensors and actuators are well suited to its environment.

2.2 Rational behaviour

If we are to construct an agent program for a given agent, what should it do? We have previously stated that we aim to construct **rational** agents; what does rationality mean?

In order to define rational behaviour, we must add a **performance measure** to our agent. A performance measure is a numerical metric that expresses the goals of an agent. For example, a web search engine's performance metric might be the number of web pages it finds that the user judges relevant.

As a general rule, a performance metric should express the agent's goal, and leave it up to the agent program to determine the best way to achieve that goal.

Given an agent, we define a rational action as the action expected to maximize its performance metric, given its history and prior knowledge. Note that we do not expect omniscience—we do not expect an agent to necessarily act perfectly, but simply require it to act as well as it can, given the knowledge it has.

The extent to which the agent uses prior knowledge (instilled in it by its designer) rather than its own experience, is the extent to which that agent lacks **autonomy**. In general, we aim to build autonomous agents because they require less expert knowledge and are more adaptable than agents that require a great deal of problem-specific knowledge. However, autonomous agents are typically much harder to design—it is much easier to build a problem-specific agent with all the knowledge that it might need than to build a general one that can acquire that knowledge.

Learning activity

Write down a performance measure for a mouse, given your previous description of a mouse as an agent.

2.3 Tasks

We can consider a combination of performance measure, environment, sensors and actuators to be a **task** which the agent must solve.

Some dimensions along which tasks may differ are:

- **Fully versus partially observable**. In a fully observable task, the current percept reveals all of the relevant state of the environment, and therefore the agent does not need to internally keep track of the world. In a partially observable task the current percept only reveals some of the relevant information.
- **Deterministic versus stochastic**. In a deterministic task, the current state of the environment and the current action are sufficient to exactly predict the next state of the environment. In a stochastic task, there may be some uncertainty, often expressed as a probability distribution, as to the next state of the environment.
- **Episodic versus sequential**. In an episodic task, the agent faces a sequence of independent tasks (or episodes). In a sequential task, the agent's next state always depends on the environment and its current state.
- **Static versus dynamic**. A static task takes place in an environment that does not change without the agent's intervention; a dynamic task takes place in an environment that may.
- **Discrete versus continuous**. A discrete task has a finite number of distinct states, percepts and actions (often represented by integers). In a continuous task, there may be infinitely many of any of these (often represented by the real numbers over an interval).
- **Known versus unknown**. When the task environment is known, the agent is able to predict the consequences of its own actions—for example, a chess playing agent knows the rules of chess. When it is unknown, the agent does not know, and must discover, how the environment 'works'.
- **Single versus multi-agent**. In a multi-agent task, other agents may exist in the environment with their own performance measure. An agent may wish to compete or cooperate with the other agents in the environment.

For example, the task of flying a helicopter from one point to another is fully observable (given speed, pose, and map location), stochastic (because of random air currents), sequential, dynamic (because weather conditions may change), continuous and single agent. The game of draughts (also known as checkers) is fully observable, deterministic, sequential, static, discrete and multi-agent.

These characteristics of the task, taken together, will determine how difficult the task is to solve, and what kinds of techniques can be applied to solve it. Therefore, a task analysis is often the first undertaking in designing an agent program to solve a particular task.

Learning activity

Consider an agent that must learn to play Backgammon by playing against itself. Describe its task using the dimensions above.

2.4 Types of agents

There are many ways to implement an agent program, depending on the environment and what the agent must accomplish. Some broad agent categories are:

- **Simple reflex agents** select the current action based only on the current percept. Such agents may be suitable for fully observable tasks.
- **Model-based reflex agents** use a model of the world to augment the information in the current percept to determine which action to take.
- **Goal-based agents** employ knowledge of the goals encoded in their performance metric and then employ planning (see Chapter 5) to select actions that will lead them to their goals.
- **Utility-based agents** map states in the environment to a measure of utility in order to select actions that lead to better performance.
- **Learning agents** use their experience in the world to improve their performance over time. Learning could be be incorporated into any of the above types of agents.

2.5 Learning outcomes

By the end of your study of this chapter, the Essential reading and activities, you should be able to:

- Break a task down into agent and environment components.
- Describe the resulting agent and environment using the terms introduced in this chapter.
- Write down a performance metric for the task.
- Suggest a suitable type of agent for solving it.

Chapter 3

Search

Essential reading

Russell, S. and P. Norvig *Artificial Intelligence: A Modern Approach*. (Upper Saddle River, NJ: Prentice Hall, c2010) third edition [ISBN 9780132071482 (pbk); 9780136042594 (hbk)]. Chapter 3.

3.1 Introduction

Earlier we saw that a task consists of an agent's sensors, actuators, environment and performance metric. Often the agent's performance metric specifies a particular **goal** (or set of goals) and a cost measure for reaching that goal.

In many cases, we can formulate the task of a goal-seeking agent as a well-defined **problem**. Given a set of environment **states**, a problem consists of four components:

1. An **initial state** (or set of initial states) that the agent must start from.
2. A set of **actions** that the agent may take for each state. This is often described by a **successor function** which maps each state s to a set of action-successor state pairs. Successor pair (s', a) indicates that taking action a from state s leads to state s'.
3. A **goal test** that indicates which states are goal states. In some cases the list of goal states is given explicitly; in others, it may be more complex to determine whether or not a given state is a goal.
4. A **path cost** function that assigns a numeric cost to each path through the state space. The agent's performance measure thus consists of reaching a goal state while at the same time minimising its path cost.

A **solution** to a problem is a path from a start state to a goal state; an **optimal solution** is any solution with minimum path cost (there may be more than one).

Consider the problem of navigating through a news website (say, the BBC) to find a news story with a particular quote that we wish to cite. In such a problem:

- The set of states are all of the webpages in the site; note that we may not know in advance how many of them there are.
- The goal test is whether or not the quote shows up on the page. Note that the quote could occur in several stories on the same topic.
- The set of actions that can be taken at any state is the set of links on that page, each of which leads to a new page.
- The path cost is one for each link (because it costs us time to access each page).

Learning activity

Write down the problem formulation of the Rubik's Cube puzzle.

3.1.1 Problem solving as search

Given a problem setting, we are tasked with designing an agent capable of finding a solution. This is a **search** problem: the agent must search for a solution out of all possible paths through the states.

We can view the search space as a tree. Nodes on the tree correspond to states in the problem, and edges in the tree correspond to taking an action from one state to another. At the first level of the tree, we have the start state; at the second level, all of its successor states, and so on. (Note that states may appear at multiple places in the tree.)

A generic search algorithm works as follows. We keep a list of **open** nodes, starting with just the start state. We then repeatedly remove a single node from the list of open nodes, and **generate** all of its successor nodes. We test whether any of the successor states are goals; if any are, we stop. If none are, we place those successor nodes on the open list, and repeat. This process is depicted in Figure 3.1.

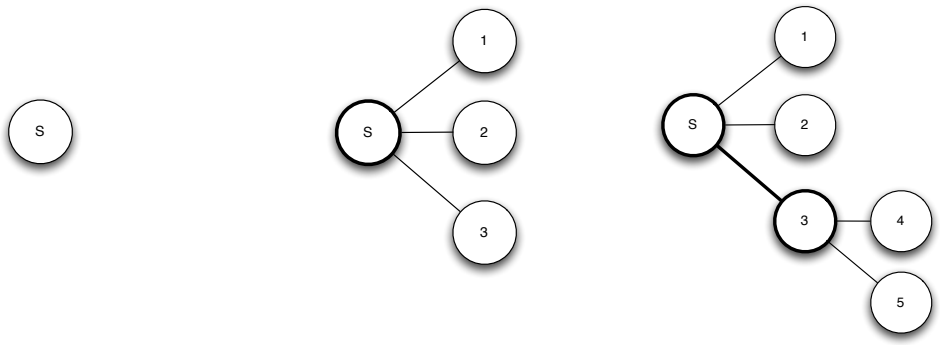


Figure 3.1: Searching through a small tree. Visited states have thick outlines; states on the open list are shown with thin outlines. At first, only the start state s is on the open list (a). It is not the goal, so its successors (states 1, 2 and 3) are generated and placed on the open list, and s is removed (b). Next, state 3 is selected from the open list, and its successors (4 and 5), also not the goal, are placed on the open list; state 3 is then removed. (c) This process continues until a goal state is found.

Differences in search methods revolve primarily around the method for selecting which of the states on the open list to expand next.

When evaluating different search methods, we consider these four criteria:

1. **Completeness**. Will the method always find a solution if one exists?
2. **Optimality**. Will the method always find the optimal solution?
3. **Time complexity**. How long does the method take?

4. **Space complexity**. How many states must be stored in memory during search?

Three quantities will be useful when measuring complexity: the **maximum branch factor** b of the search tree; the **minimum depth** of a solution in a tree, d; and the **maximum path length**, m. These are depicted in Figure 3.2.

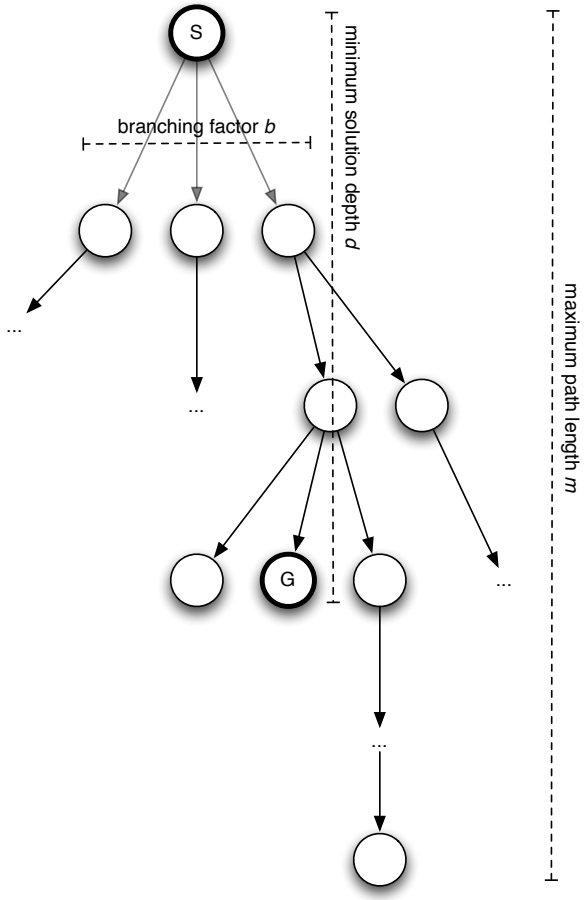


Figure 3.2: Quantities useful in analysing the complexity of searches. S is the start state, and G is the goal state. Omitted branches indicated by '...' in the diagram.

3.2 Uninformed search methods

In this section we consider search methods that are applicable when the agent is only given the problem setting: it has no further knowledge of the problem, and therefore no hints as to which states might be better than others.

3.2.1 Breadth-first search

Breadth-first search expands the state in the open list with the minimum depth in the tree. Thus, all states at depth k will be expanded before any of the states at depth

$k + 1$. Every node in a level of the tree is fully expanded before the next level is considered. An example search tree is shown in Figure 3.3.

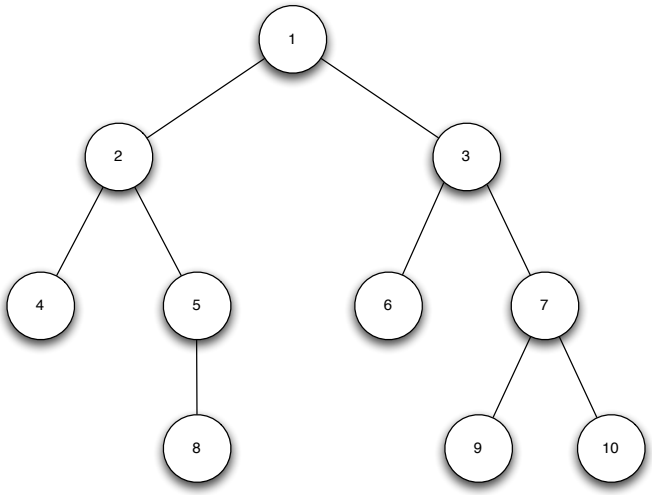


Figure 3.3: An example search tree, showing the order of expansion for the nodes visited by a Breadth-first search.

Provided the branching factor b is finite, breadth-first search will always find a solution and is therefore complete. However, it finds the **shortest** path to the goal, and is therefore not necessarily optimal—there may be longer paths of lower cost. A small change—selecting the state with the smallest total path cost rather than depth at each step—results in a **uniform cost search**.

The time and space complexity for Breadth-first search is $O(b^{d+1})$. Since this cost is exponential in the depth of the search tree d, for all but the smallest problems Breadth-first search is completely infeasible.

3.2.2 Depth-first search

In contrast, Depth-first search always expands the state in the open list with the maximum depth. Thus, the search proceeds through the first branch of the tree all the way down to the bottom, before exploring the second branch. An example search tree is shown in Figure 3.4.

Because Depth-first search explores to the leaves of the search tree and does not keep whole slices of the tree in memory, it has an $O(bm)$ space complexity, which is a significant improvement on breadth-first search. However, since it may explore entire paths through the tree while missing a much shorter solution, it has worst-case time complexity $O(b^m)$. It is complete if b and m are finite, but even in that case it is not optimal. It is common for b to be finite, but m is often not finite.

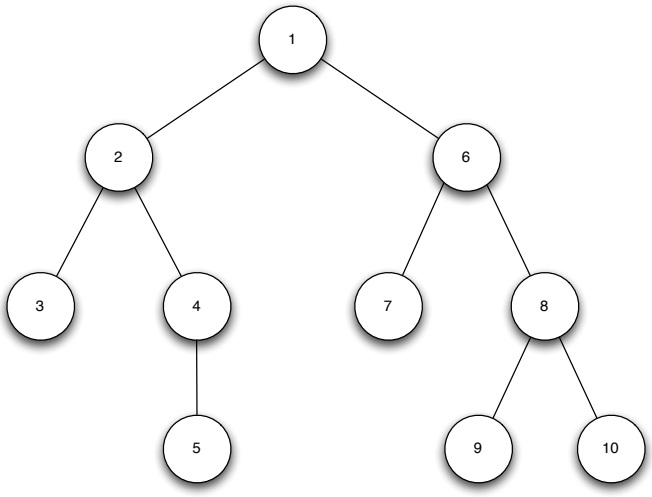


Figure 3.4: An example search tree, showing the order of expansion for the nodes visited by a depth-first search.

3.2.3 Iterative deepening search

Iterative deepening search combines depth-first and breadth-first searches to obtain the small memory footprint of depth-first search with the completeness and optimality properties of breadth-first search.

An iterative deepening search treats the search problem as a sequence of depth-first searches, each with an increasing limit on the depth of the tree. Thus, the first search limits the tree to depth 1, the second to depth 2, and so on.

Note that this has the effect of repeating searches: once the tree has been searched to depth k, it must be re-searched to depth $k + 1$. Although this seems wasteful it is not that expensive: because the number of nodes at each level is b times the number of nodes at the previous level, the number of nodes at level l increases exponentially with l. Thus, for reasonably deep trees, the cost of expanding an extra level dwarfs that of re-searching earlier levels.

Indeed, iterative deepening search has a time complexity of $O(b^d)$, which is faster than breadth-first search, and it has a space complexity of only $O(bd)$, lower than both depth-first and breadth-first search. Like breadth-first, it is complete when b is finite, and finds the shortest path to the goal (although not necessarily the optimal, least-cost path).

Learning activity

In the above definitions, we have simply stated the time complexities of each search algorithm. Explain convincingly (or preferably prove) why these complexities hold.

3.3 Informed search methods

In the previous section, we considered search problems where our agents know nothing about the problem other than its formulation. However, in many cases we **do** know something about the problem.

For example, in the introduction to this chapter we considered the example of an agent searching through the BBC news website for a web page with a quote on it. If we know that the quote is from a politician, then it would seem sensible to search the Politics section of the website before the Entertainment section.

An **informed search** uses an **heuristic**, or rule-of-thumb metric, that tells us something about how close a state is to the goal, to order the expansion of nodes during search. Below we consider one such method, A* search.

3.3.1 A* search

In A* search, we assume that we are given a heuristic function h that, for any state, estimates the distance from that state to the goal. When selecting states in the open list to expand, we pick the state s that minimizes $f(s) = h(s) + g(s)$, where $g(s)$ was the total path cost to reach s.

$f(s)$ gives us a measure of estimated total path cost: it is the estimated cost from s to the goal (as given by our heuristic $h(s)$) plus the cost to reach s from the start state (as given by $g(s)$). Thus, A* expands states in order of least estimated total path cost.

While $g(s)$ can be easily computed, $h(s)$ is an estimate. Thus, the quality of our search depends on the quality of our heuristic. It turns out that A* is both complete and optimal if h is an **admissible heuristic**—that is, $h(s)$ never overestimates the cost to the goal.

For example, consider the problem of searching for the shortest way to travel between two distant cities, using the train. Since we know that the shortest distance between two points is a straight line, we might set $h(s)$ for city s to be the straight-line distance between city s and the goal city. Since this is always less than or equal to the real distance travelled by the train (which may take a winding course around natural obstacles or through other cities), it is an admissible heuristic for the shortest rail route problem.

Learning activity

Suggest a heuristic for Rubik's Cube. Is it admissible? Support your answer.

3.4 Exercises

1. Consider Sudoku. Write down a problem formulation for it. What kind of search method would you use to solve it?

2. Write down the complexity of your chosen search method. What are the values of b, m and d for your Sudoku problem formulation? Approximately how many states would your search need to visit, and how many would it keep in memory, in the worst case?
3. Prove that the solution found by A* with an admissible heuristic is optimal. Hint: write down what you know about an admissible heuristic, and use a proof by contradiction.

3.5 Learning outcomes

By the end of your study of this chapter, the Essential reading and activities, you should be able to:

- Formulate a well-defined problem description from a textual description of a new task.
- Determine which search algorithm should be applied to a new problem, and explain why.
- Define a heuristic if necessary, and prove that it is admissible (or show that it is not).
- Determine the worst-case time complexity of your approach.

Chapter 4

Knowledge representation and reasoning

Essential reading

Russell, S. and P. Norvig *Artificial Intelligence: A Modern Approach*. (Upper Saddle River, NJ: Prentice Hall, c2010) third edition [ISBN 9780132071482 (pbk); 9780136042594 (hbk)]. Chapters 7 and 8. These two chapters introduce knowledge representation and basic reasoning using propositional and first-order logic.

Russell, S. and P. Norvig *Artificial Intelligence: A Modern Approach*. (Upper Saddle River, NJ: Prentice Hall, c2010) third edition [ISBN 9780132071482 (pbk); 9780136042594 (hbk)]. Chapter 13. This chapter outlines the basics of probability theory and Bayesian inference.

Further reading

Russell, S. and P. Norvig *Artificial Intelligence: A Modern Approach*. (Upper Saddle River, NJ: Prentice Hall, c2010) third edition [ISBN 9780132071482 (pbk); 9780136042594 (hbk)]. Chapter 9. Describes inference in first-order logic in detail.

Russell, S. and P. Norvig *Artificial Intelligence: A Modern Approach*. (Upper Saddle River, NJ: Prentice Hall, c2010) third edition [ISBN 9780132071482 (pbk); 9780136042594 (hbk)]. Chapter 14. Describes methods for probabilistic reasoning.

Mellish, C.S. and W.F. Clocksin *Programming in PROLOG: Using the ISO Standard.* (Springer, 2003) fifth edition [ISBN 9783540006788]. A good introduction to *PROLOG*, a declarative programming language based on first-order logic.

Bertsekas, D.P. and J.N. Tsitsikilis *Introduction to Probability.* (Athena Scientific, 2002) [ISBN 188652940X]; also (Athena Scientific, 2008) second edition [ISBN 9781886529236 (hbk)]. An excellent and very clear introduction to probability.

4.1 Introduction

One important aspect of intelligence is the ability to **represent knowledge** about the world internally, and to use it to draw further conclusions about the world through **reasoning**.

An agent that can keep track of what it knows and use that information to infer new knowledge about the world is potentially very flexible and general. Such an agent is called a **knowledge based agent**.

A knowledge based agent usually maintains a collection of facts and rules, called a **knowledge base**, to which it can add new facts and rules, and from which it can derive new facts and rules. A knowledge base is usually constructed using some form of **representation language**, coupled with an **inference procedure** for determining the truth of conjectured facts.

Programming using a representation language is a form of **declarative programming**, where information is declared (presented as facts) and desired outputs are specified as questions about the truth of a fact, leaving the details of determining the answers to the inference procedure. Thus the programmer is concerned primarily with **what** is to be achieved. This can be contrasted with regular **procedural programming**, in which the programmer is concerned with **how** a specified output is to be obtained.

For example, a programmer using a procedural programming language to find the solution to a Sudoku puzzle would spend their time writing code expressing how to conduct the search for a solution (for example, implementing a breadth-first search). In contrast, a programmer using a declarative language would describe the start state of the puzzle and the conditions that must be fulfilled by a solution (that each digit occurs exactly once in a row, column and block). The programming language implementation would then use the declared information to search for a solution that fulfills the requested conditions automatically.

The form of the representation language dictates what kind of knowledge can be represented, and what kind of inference procedure can be practically applied to the knowledge base. We consider a few example languages in the following sections.

4.2 Propositional logic

Propositional logic is a simple logic that deals with symbols and their relationships with each other. This allows us to create agents that deal with known facts, and can use these to make decisions by reasoning logically about what is known.

Each symbol (or **proposition**) is either a variable (denoted by an uppercase letter, e.g. A) or a truth literal: *True* or *False*, and can be used to build up **sentences** in propositional logic. Any single symbol or literal is a sentence by itself. Furthermore, if A and B are sentences, then any of the following are also sentences:

Sentence	Semantics
$\neg A$	A is *False*. Read as *not A*. Note that $\neg\neg A$ means A is *True*.
$A \vee B$	One or both of A and B is *True*. Read as *A or B*.
$A \wedge B$	Both A and B are *True*. Read as *A and B*.
$A \Rightarrow B$	If A is *True*, then so is B. Read as *A implies B*. Note that if A is *False* then B may be either *True* or *False*.
$A \Leftrightarrow B$	A and B have the same truth value. Read as *A if and only if B*.

Any sentence produced this way can also naturally be used to generate further sentences. For example, the following are all sentences in propositional logic:

- $C \wedge B \Rightarrow D$
- $(C \Leftrightarrow D) \vee E$
- $((\neg X) \wedge B) \vee (Y \wedge X)$

(Although it is not strictly correct syntax, in practice we use brackets to make operator precedence clear.)

Note that any sentence generated this way is **well-formed, but not necessarily true**. In addition, the sentences that are used to generate a true sentence may not be true. For example, both A and $\neg A$ are well-formed sentences, but in a consistent knowledge base both cannot be true. Thus, the description of which sentences are well-formed is a description of the language **syntax**.

4.2.1 Entailment

A knowledge base in propositional logic consists of a set of valid sentences that are asserted to be true. The task of the agent is then to make decisions by posing queries to its knowledge base. The answers to the queries are usually not immediately present in the knowledge base, but must be inferred.

A **model** is an assignment of truth values to all of the variables in a knowledge base. A set of sentences is said to be **consistent** with a model if every sentence in the set is *True* in that model. (Note that many models may be consistent with a set of sentences.) Given a set of sentences P, we say that P **entails** sentence B (written $P \models B$) if, for every possible model in which P is consistent, B is *True*. In other words, $P \models B$ if the sentences in P imply that B is *True*.

For example, consider an agent concerned with predicting the weather, with variable R indicating whether or not it is likely to rain, variable C indicating whether or not it is cloudy, and variable L indicating low pressure. Given knowledge base K:

- L *(Pressure is low)*
- C *(It is cloudy)*
- $C \wedge L \Rightarrow R$, *(Clouds and low pressure imply rain)*

the agent may conclude R; thus, the agent's knowledge implies that R is true, because $K \models R$.

Similarly, given knowledge base L:

- $\neg L$ *(Pressure is high)*
- C *(It is cloudy)*
- $C \wedge L \Rightarrow R$, *(Clouds and low pressure imply rain)*

the agent cannot conclude that R is true; $L \not\models R$. **Note that this does not mean R is False; it simply means that the agent cannot necessarily conclude it with the knowledge it has**. Another way of saying this is that there are models that are consistent with the knowledge base but in which R is *False*.

Learning activity

Show that the above two conclusions are correct using a *truth table* of all possible values for C, L and R; each column should correspond to a variable and each row a truth assignment to every variable. Eliminate rows where the sentences in the knowledge base do not all hold, and examine the value of R in the remaining rows.

Adding a sentence to the knowledge base expressed in propositional logic must increase (or keep the same) the number of sentences entailed by it. This is known as the **monotonicity** property. Some logical formalisms allow **nonmonotonic reasoning**: they allow the addition of a sentence to a knowledge base to cause sentences that were previously entailed by the knowledge base to no longer be entailed.

4.3 Reasoning using propositional logic

So far we have shown how knowledge can be represented using propositional logic, and a few examples of how new knowledge can be inferred from an existing knowledge base. However, in order to build knowledge-based agents we must have formal reasoning processes for performing inference.

Such a process should, given a **goal sentence**, try to show that it is true by producing a **proof** (a sequence of inference steps leading from the sentences in the knowledge base to the goal sentence). Two important properties that we aim for in reasoning processes are **soundness** (the resulting sentences are actually entailed by the knowledge base) and **completeness** (any sentence entailed by the knowledge base can be inferred).

The most obvious reasoning method is **enumeration**, where we examine all combinations of truth assignments for every variable in the knowledge base. For every truth assignment we can then evaluate whether the knowledge base is consistent (by checking that all statements in it are true); if it is, then the goal sentence should also be true. Any variable assignment in which the knowledge base is consistent but the goal sentence is false is a contradiction; if there are none, then the exhaustive list of assignments in which the knowledge base is consistent and the sentence is *True* constitutes a proof.

Learning activity

How many rows are there in a truth table for n variables?

4.3.1 Reasoning patterns

The enumeration method becomes very slow for large numbers of variables, and in general we would prefer more efficient methods. These methods are primarily based on using **logical equivalences**—transformations from one sentence (or set of sentences) to another sentence that is true in the same set of models. These equivalences can be used as actions in a search method as described in Chapter 3. Figure 4.1 shows the standard set of logic equivalences for propositional logic.

Learning activity

Show that De Morgan's Laws are true using a truth table.

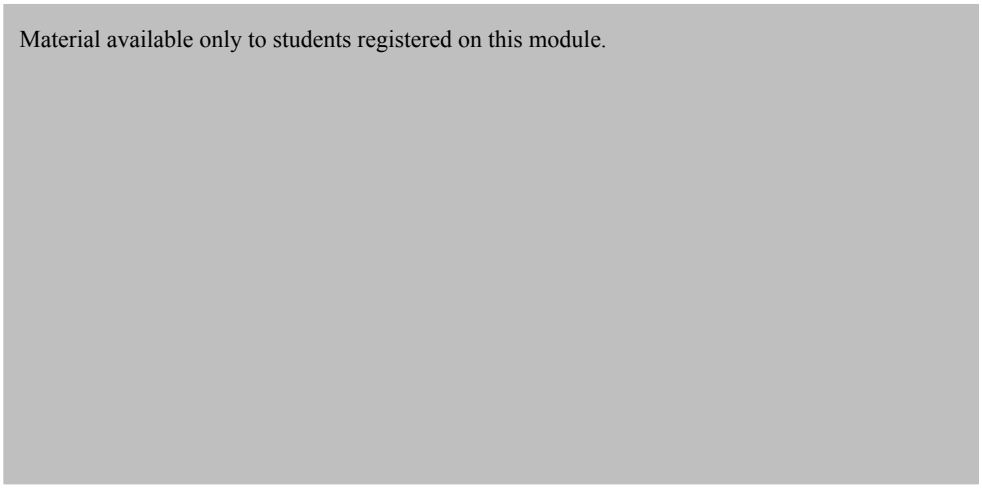


Figure 4.1: Standard logic equivalences, from Russell and Norvig, Chapter 7, page 249. A, B and C are sentences in propositional logic. (RUSSELL, STUART; NORVIG, PETER, *ARTIFICIAL INTELLIGENCE: A MODERN APPROACH*, 3rd edition, ©2010. Electronically reproduced by permission of Pearson Education, Inc., Upper Saddle River, New Jersey.)

Each of these rules can be used to produce a **reasoning pattern**, which specifies a sound way to manipulate existing sentences to obtain new, logically equivalent, sentences. Reasoning patterns are written in the following format:

$$\frac{A}{B},$$

where A is a set of logical sentences that must be given, and B is the logical sentence that can then be inferred. For example, De Morgan's first law can be used to obtain the following reasoning pattern:

$$\frac{\neg(A \wedge B)}{(\neg A \vee \neg B)}.$$

Other common reasoning patterns are **and-elimination**:

$$\frac{A \wedge B}{A},$$

which allows us to conclude that each of a set of terms that have been $\wedge$ed together must be true, and **modus ponens**:

$$\frac{A \Rightarrow B, \quad A}{B},$$

which allows us to complete an implication if its condition is true. An especially important reasoning pattern is the **unit resolution** rule:

$$\frac{l_1 \vee l_2 \vee \ldots \vee l_k, \quad \neg l_i}{l_1 \vee l_2 \vee \ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k},$$

which allows us to remove false literals from $\vee$ statements. All of these rules are sound; a slightly more general version of the resolution rule is complete by itself when coupled with a complete search procedure.

As an example of the application of reasoning patterns, consider the following

knowledge base:

1. $\neg A \wedge B \Rightarrow C$
2. $C \vee B$
3. $\neg(C \wedge D)$
4. D

We may now query the truth of the logic sentence A. We first apply De Morgan's rule to fact 3:

$$\frac{\neg(C \wedge D)}{\neg C \vee \neg D},$$

resulting in sentence 5: $\neg C \vee \neg D$.

We follow this by applying unit resolution to sentences 4 and 5:

$$\frac{\neg C \vee \neg D, \quad D}{\neg C},$$

resulting in sentence 6: $\neg C$.

We then apply contraposition to sentence 1:

$$\frac{\neg A \wedge B \Rightarrow C}{\neg C \Rightarrow \neg(\neg A \wedge B)},$$

obtaining sentence 7: $\neg C \Rightarrow \neg(\neg A \wedge B)$.

Since from sentence 6 we know $\neg C$, we can apply modus ponens to sentence 7:

$$\frac{\neg C \Rightarrow \neg(\neg A \wedge B), \quad \neg C}{\neg(\neg A \wedge B)},$$

resulting in sentence 8: $\neg(\neg A \wedge B)$.

Applying De Morgan's again:

$$\frac{\neg(\neg A \wedge B)}{\neg\neg A \vee \neg B},$$

sentence 9: $\neg\neg A \vee \neg B$.

Applying unit resolution to sentences 6 and 2:

$$\frac{B \vee C, \quad \neg C}{B},$$

resulting in sentence 10: B.

Unit resolution again, for sentences 9 and 10:

$$\frac{\neg\neg A \vee \neg B, \quad B}{\neg\neg A},$$

sentence 11: $\neg\neg A$.

Finally, we apply double-negation elimination to sentence 11:

$\frac{\neg\neg A}{A}$, resulting in sentence 12,

our goal sentence, A.

Thus, our knowledge base entails A, and we may return *True*.

Learning activity

Given the same knowledge base, and without referring to the working above, determine the result of the query B.

4.4 First-order logic

Although propositional logic is a useful representational tool for very simple problems, it lacks the expressive power that would make it suitable for larger scale problems.

For example, given variable A (which indicates that a particular object is an apple), it is possible to represent the fact that A being true implies that the apple is either green or red, by having one variable for it being red, and one for it being green. However, if there were many apples, including such a rule and pair of extra variables for each apple explicitly would be cumbersome. It would be preferable if there was a mechanism for expressing the notion that *all apples are either red or green*.

First-order logic is a representational language that deals with **objects** and **relations** between one or more objects that may or may not hold. It is a powerful way to express knowledge that allows for concise knowledge representation.

4.4.1 Symbols

In propositional logic there is only one kind of symbol: propositions (e.g., A) that represent facts about the world, and can be either *True* or *False*. In first-order logic, there are instead three kinds of symbols:

1. **Constants**, which represent objects in the world. For example, A could be an object in the world. Note that A does not have a *True* or *False* value; it simply refers to something in the world.
2. **Predicates**, which stand for relations between one or more objects in the world, and have a *True* or *False* value; for example, *Apple(A)* or *Colour(A, Red)*. A predicate can be thought of as a list of sets of objects. For example, the predicate *Colour* has two arguments, and so can be thought of as a list of pairs of objects where the *Colour* relation holds. Its truth value for a given pair of objects then indicates whether or not that particular pair of objects is in the list.
3. **Functions**, which take a set of objects as input and return another object. For example, *Colour(A)* might return the symbol *Red*.

First-order logic also includes all of the connectives available in propositional logic, so we can form sentences like:

- $Apple(A) \Rightarrow Colour(A, Red) \vee Colour(A, Green)$
- $Pressure(Weather, Low) \wedge Cloudy(Weather) \Rightarrow Rainy(Weather)$

4.4.2 Quantifiers

If we were to use first-order logic as defined so far to represent knowledge about apples, we would still have to go through all possible objects for which *Apple* is true and write a rule about their colour. However, first-order logic includes two extra operators, called **quantifiers**, which allow us to write much more general statements.

The **universal quantifier** $\forall$ (read: *for all*) introduces a variable and then asserts that, for any symbol that can be bound to that variable, a following statement holds. For example, if we wished to assert that *for all apples x, x is red or green* we could say:

$$\forall x, \mathit{Apple}(x) \Rightarrow \mathit{Colour}(x, \mathit{Red}) \vee \mathit{Colour}(x, \mathit{Green}),$$

which is read as 'for all x, if x is an apple then x is red or green', or summarised as 'apples are red or green'.

So, if we add that statement to our knowledge base, we are asserting that all apples are green or red. If we add a new object and state that it is an apple, we can automatically infer that its colour is red or green without any further statements.

We can also use such a statement as a **question** to the knowledge base, to ask: *is it the case that all apples in the knowledge base are known to be red or green?*

The **existential quantifier** $\exists$ (read: *there exists*) introduces a variable and asserts that **at least one** object can be bound to it such that a following statement holds. For example, if we wished to assert that *at least one apple has spots* we can write:

$$\exists x, \mathit{Apple}(x) \wedge \mathit{Spots}(x),$$

which can be read as 'there exists an x such that x is an apple and x has spots', or summarised as 'there exists an apple with spots'.

Adding that statement to our knowledge base asserts that at least one of the apples we know of has spots (without specifying which one). If we use it as a question, then we are asking: *is at least one apple in the knowledge base known to have spots?*

Learning activity

Write a propositional logic sentence expressing the following: *red apples and also large green apples are tasty*.

Because an existential statement is quite weak while a universal statement is very strong, it is common for statements in the knowledge base to be universal statements (e.g. 'all birds have beaks'), whereas queries to the knowledge base are often existential statements (e.g. 'is there an animal with four legs?').

Learning activity

Would you expect an existential or a universal query to take longer to answer for a large knowledge base? Why?

4.4.3 Inference in first-order logic

Effective inference in first-order logic is a complex subject beyond the scope of this subject guide. However, two important inference rules that deal with quantifiers will illustrate some aspects of it.

The first rule deals with universal quantifiers, and is called **universal instantiation**. Given a universally quantified expression such as:

$\forall x\ A(x)$,

where $A(x)$ is some logical expression using quantifier x. We may **instantiate** this rule by picking a constant symbol and substituting it for x. Thus, if we have constant symbol B we may conclude $A(B)$. We are free to choose any (and all) constants that exist in the Knowledge Base, and so may apply a heuristic to decide which to use.

The situation is slightly more complex for existential quantifiers. In **existential instantiation** we may create a **new** constant symbol for an existentially quantified rule. For example, given rule:

$\exists x\ C(x)$,

we may introduce a new constant D and conclude $C(D)$. Note that D may in fact be equal to existing constant in the knowledge base, or it may not.

Learning activity

Suggest a heuristic for selecting which symbols to try first when instantiating a universally quantified expression, given a query sentence.

4.4.4 An example knowledge base

Consider the following knowledge base. First, we have a set of objects, in this case pieces of food:

1. *Apple(A)*
2. *Apple(B)*
3. *Orange(C)*
4. *Pear(D)*
5. *Carrot(E)*
6. *Onion(F)*
7. *Pepper(G)*

We also have some general rules about fruits and vegetables:

1. $\forall x$, $Fruit(x) \vee Vegetable(x)$ *(everything is a fruit or a vegetable)*
2. $\forall x$, $Apple(x) \Rightarrow Red(x) \vee Green(x)$ *(apples are red or green)*
3. $\forall x$, $Apple(x) \vee Orange(x) \vee Pear(x) \Rightarrow Fruit(x)$ *(apples, oranges and pears are fruit)*
4. $\forall x$, $Carrot(x) \vee Pepper(x) \Rightarrow Vegetable(x)$ *(carrots and peppers are vegetables)*

5. $\forall x$, *Fruit(x)* $\Rightarrow$ *Tasty(x)* *(all fruit are tasty)*
6. $\forall x$, *Carrot(x)* $\Rightarrow$ *Tasty(x)* *(all carrots are tasty)*
7. $\forall x$, *Onion(x)* $\Rightarrow \neg$ *Tasty(x)* *(onions are not tasty)*

We may now pose the following query to the knowledge base:

Fruit(A)?

to which we expect a response of *True*. To compute such a response, we can instantiate rule 3 using constant A to obtain:

$$Apple(A) \vee Orange(A) \vee Pear(A) \Rightarrow Fruit(A),$$

and then apply modus ponens to it given fact 1 to obtain:

$$\frac{Apple(A) \vee Orange(A) \vee Pear(A) \Rightarrow Fruit(A), \quad Apple(A)}{Fruit(A)},$$

which results in *Fruit(A)*.

Another example query:

Vegetable(F)?

Since we know that $Onion(F)$, we may instantiate rule 7 ($\forall x$, *Onion(x)* $\Rightarrow \neg$ *Tasty(x)*) using the symbol F to conclude $\neg Tasty(F)$. We can also then instantiate rule 5 with the symbol F to obtain *Fruit(F)* $\Rightarrow$ *Tasty(F)*. Using these two new pieces of information, we may apply contraposition:

$$\frac{Fruit(F) \Rightarrow Tasty(F)}{\neg Tasty(F) \Rightarrow \neg Fruit(F)},$$

and then applying modus ponens:

$\neg Fruit(F)$.

Instantiation rule 1 ($\forall x$, *Fruit(x)* $\vee$ *Vegetable(x)*) using constant F and applying resolution, we obtain *Vegetable(F)* and find the query *True*.

However, the query

$Fruit(E)$?

should return *False*, even though carrots are tasty—because while it is true that carrots are tasty (rule 6) and all fruit are tasty (rule 5), it does not follow that objects that are **not** fruit are not tasty. The situation would be different if the connective in rule 5 was $\Leftrightarrow$ instead of $\Rightarrow$.

Moreover, of the following queries:

- $\exists x$, *Red(x)?*
- $\exists x$, *Green(x)?*
- $\exists x$, *Green(x)* $\vee$ *Red(x)?*

only the last can be found *True*, because although we know that all apples are either red or green, we do not have any further information, and so cannot know whether

they are all green, all red, or mixed.

4.5 Uncertain knowledge

The use of logic as described above makes an important commitment: that the information the agent adds to its knowledge base is completely certain, and so inferences drawn from it are also completely certain. However, when interacting with real world domains, we often find that we are dealing with uncertain knowledge; moreover, we often find that inferences are not always perfect.

For example, consider the thought processes of a doctor diagnosing a patient. The presence of a particular symptom may suggest a disease, but in most cases it does not indicate it with perfect certainty. In addition, the instruments used to measure those symptoms are not perfect: sometimes they return false positives, or false negatives.

Learning activity

What is a false negative? What is a false positive?

4.5.1 Probability theory

Fortunately, we already have a powerful tool for handling reasoning with uncertainty: probability theory provides a sound, principled method for decision making under uncertainty. In the remainder of this section, we will briefly examine how the rules of probability can be used as a knowledge representation language.

In propositional logic, we used variables that could take on the logical values *True* or *False*. In probability, we use **random variables** instead. Random variable A has two aspects: the set of values that it can be assigned (its **domain**), and a probability in the range $[0, 1]$ for each possible assignment, indicating the probability that the variable has that particular value. There are three types of random variables:

1. **Boolean random variables**, that can take on values *True* or *False*.
2. **Discrete random variables**, that can take on a finite number of values.
3. **Continuous random variables**, that can take on an infinite number of values. Continuous random variables most commonly have a domain that is some segment of the real number line. For example, the variable *Depth* which might range anywhere from 0 to 1 metres.

In this section we will deal only with boolean random variables, since they are the simplest case.

Given random variable A, we may express the probability that $A = True$ by $P(A)$, and the probability that it is false by $P(\neg A)$. The collection of probabilities for all possible random variable assignments is called a **probability distribution**.

The following three axioms underlie probability theory:

1. $P(True) = 1$; $P(False) = 0$. True variables are certain to be true; False variables are certain to not be true.
2. $0 \leq P(A) \leq 1$. The probability of any variable being true lies between or at 0 (impossibility) and 1 (certainty).
3. $P(A \vee B) = P(A) + P(B) - P(A \wedge B)$. The probability of at least one of two variables being true is the sum of the probabilities of either one being true, minus the probability of them both being true.

In addition, we can derive the following important rule from the axioms above:

- $P(A \vee \neg A) = P(A) + P(\neg A) = 1$. A boolean variable is certainly either *True* or *False*.

When we consider the quantity $P(A)$, we are considering the probability of A being true without reference to the other variables in the knowledge base. However, it may be the case that $P(A)$ actually depends on the values of other variables in the system. Consider the following table:

P	A	$\neg A$
B	0.3	0.2
$\neg B$	0.15	0.35

Figure 4.2: Joint probability distribution for example random variables A and B.

This is known as a **joint probability distribution**, and it indicates the probabilities of **pairs** (or, more generally, sets) of variables taking on particular values. Note that, as always in probability distributions:

$$P(A, B) + P(A, \neg B) + P(\neg A, B) + P(\neg A, \neg B) = 0.3 + 0.2 + 0.15 + 0.35 = 1.$$

What if we wished to know $P(A)$, the probability of A being true, without reference to B?

Notice that $P(A, B) = 0.3$, while $P(A, \neg B) = 0.15$. This indicates that the probability of A being *True* depends on the value of B.

If instead $P(A, \neg B) = P(A, B)$ then A and B are said to be **independent variables**—the probability of one variable being a particular value does not depend on the value of the other. If two variables are independent, their joint probability breaks down into individual probability distributions: $P(A, B) = P(A)P(B)$.

Learning activity

Modify the table in Figure 4.2 so that A is independent of B (making sure that the rows and columns all still sum to 1). Is B now also independent of A?

In the case where the variables are dependent, in order to take the effects of B into account we must perform a procedure called **marginalising** out B:

- $P(A) = P(A, B) + P(A, \neg B)$.

Marginalising over a variable is thus simply summing over all possible assignments of that variable.

A similar method is called **conditioning**:

- $P(A) = \sum_b P(A|b)P(b) = P(A|B)P(B) + P(A|\neg B)P(\neg B)$

where $P(A|B)$ (read as: the probability of A *given* B) indicates the probability of A given that B is *True*. This method sums the probabilities of A given each potential value of B, weighting each term in the summation by the probability of B taking that value.

In order to carry out the conditioning above, we must know three things, all of which we can obtain given the joint probability distribution:

- $P(B) = P(B, A) + P(B, \neg A) = 0.3 + 0.2 = 0.5$; further $P(\neg B) = 1 - P(B) = 0.5$.
- $P(A|B) = \frac{P(A,B)}{P(A,B)+P(\neg A,B)} = \frac{0.3}{0.3 + 0.2} = 0.6$.
- Similarly, $P(A|\neg B) = \frac{P(A,\neg B)}{P(A,\neg B)+P(\neg A,\neg B)} = \frac{0.15}{0.15 + 0.35} = 0.3$.

For the last two, the denominator in the fraction is simply a normalising constant to make sure that the resulting probability is properly scaled. Otherwise, $P(A|B) + P(\neg A|B)$ would not sum to 1.

When conditioning, we note that A is independent of B if $P(A|B) = P(A)$, because knowing the value of B does not give us any information about the value of A.

Learning activity

Determine $P(B|A)$, given the table in Figure 4.2.

4.5.2 Bayes' rule and probabilistic inference

We may ask why we should go to all the trouble of conditioning (or even marginalising) when a joint probability distribution contains all the information we might want to know. However, in real applications the full joint probability table is much too large to store in memory (or fill in a reasonable time).

Learning activity

Given n variables, how many entries are there in the joint probability table for the variables if they are all dependent on each other? How many unique entries are there if they are all independent of each other?

Instead, we often have pieces of knowledge about the world that could be used to infer parts of the table when necessary. For example, we might know the general likelihood of a variable ($P(A)$) without knowing how it interacts with other variables; or, we might know how a particular event changes the probability of another without knowing the full table. In such cases (which account for the vast majority of probabilistic AI systems), we can perform inference by making use of Bayes' rule:

- $P(A|B) = \dfrac{P(B|A)P(A)}{P(B)}$.

Bayes' rule allows us to infer the **posterior** probability of a variable A given prior knowledge about it in the form of its marginal probability ($P(A)$) and knowledge of how likely B is to be *True*, given that A is *True* ($P(B|A)$).

Thus, if we are given evidence B and asked to infer the probability of A, we can do so using the marginal probability of A (how likely A is in general), along with the probability of B if A were true.

Consider the case of a mechanic fixing a car. The mechanic observes a problem with the car: the engine is overheating, which happens in about 40 per cent of the cars she sees. The mechanic suspects that a loss of coolant is to blame; such a loss would cause the observed problem 70 per cent of the time. In addition, she knows that 20 per cent of broken down cars are due to a loss of coolant.

We can phrase this knowledge as follows, letting A be that there is a loss of coolant, and B be that the engine is overheating:

- $P(A) = 0.2$ *(prior probability of a loss of coolant)*
- $P(B|A) = 0.7$ *(loss of coolant causes overheating in* 70 *per cent of cases)*
- $P(B) = 0.4$ *(prior probability of overheating)*

The mechanic then wishes to know $P(A|B)$—*what is the probability that a loss of coolant is to blame, given that the car is overheating?* Applying Bayes' rule:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} = \frac{0.7 \times 0.2}{0.4} = 0.35,$$

indicating that there is only a 35 per cent chance that the car has lost coolant. While this result might seem surprising, it holds because there is a low prior on a loss of coolant ($P(A)$): 80 per cent of broken down cars have something else at fault.

Note that while we need to know all three quantities—$P(A)$, $P(B|A)$, and $P(B)$—to obtain a probability, if we are simply using Bayes' rule to make a decision as to whether $P(A|B)$ or $P(\neg A|B)$ is more likely, we may discard $P(B)$, since it occurs in both expressions.

The power behind Bayes' rule is that it provides us with a principled, probabilistic way to perform both inference and **belief revision**, without knowing or maintaining the full joint probability distribution.

An agent may start with a set of prior beliefs about the world, and then update them as evidence comes to light. For example, if the mechanic observed further problems with the car that might be caused by a loss of coolant, she could update her belief that it was the underlying cause. Because conditional probabilities are relatively straightforward to understand (as analogous as implications in logic), and because we can in many cases perform efficient belief updates using Bayes' rule, probabilistic knowledge representation is now the dominant approach to building knowledge representation and reasoning systems.

4.6 Exercises

1. If there were only a finite set of symbols in a knowledge base, could you convert it from first-order logic to propositional logic? How much larger would it be?
2. Could you convert a knowledge base in propositional logic to one in a probabilistic setting?

4.7 Learning outcomes

By the end of this chapter, and the Essential reading and activities, you should be able to:

- Represent knowledge using propositional and first-order logic.
- Build a knowledge base from the textual description of a problem domain.
- Apply an inference procedure to determine whether a given statement is entailed by a given knowledge base.
- Decide which question statements to ask a knowledge base in order to solve a problem in a given domain.
- Represent uncertain knowledge using probability theory.
- Perform simple probabilistic inference using Bayes' rule.

Chapter 5

Planning

Essential reading

Russell, S. and P. Norvig *Artificial Intelligence: A Modern Approach*. (Upper Saddle River, NJ: Prentice Hall, c2010) third edition [ISBN 9780132071482 (pbk); 9780136042594 (hbk)]. Chapter 10. Introduces classical planning and the PDDL planning representation language.

Further reading

Russell, S. and P. Norvig *Artificial Intelligence: A Modern Approach*. (Upper Saddle River, NJ: Prentice Hall, c2010) third edition [ISBN 9780132071482 (pbk); 9780136042594 (hbk)]. Chapter 11. Considers more realistic settings.

5.1 Introduction

The planning problem is one of determining a course of action (a series of actions) leading from some starting state to a goal state. As such, it is similar to the search problem. However, in planning we seek to formulate the problem using a representation language that exposes its structure, and employ methods that use that structure to make solving very large problems feasible.

Instead of simply having a black-box goal test, in planning the goal is explicitly represented using a logical expression; in addition, actions are not listed explicitly but instead defined compactly using logical preconditions, postconditions, and effects. Since the planner knows the structure and semantics of both expressions, it can make use of them to significantly improve the search process.

5.2 PDDL

PDDL (the **planning domain definition language**) is a classical representation language commonly used for representing planning problems, using a subset of first-order logic. We show below how PDDL is used to represent states, goals, and actions.

States are represented in PDDL as a conjunction of **positive literal predicates**. A positive literal predicate is one which does not contain a negative term (¬) and which is applied only to **ground literals** (not variables or functions). For example, if we were modelling a furniture assembly problem we may write:

- $In(PartA, SlotB) \wedge On(PartC, Table)$.

A state is thus represented as a list of predicates which are true in that state. We do not require negative predicates (those with a ¬) because predicates not mentioned are assumed to be *False*.

Goals are similarly represented as a conjunction of predicates, for example:

- $Attached(Leg1, Base) \wedge Attached(Leg2, Base) \wedge Stable(Base) \wedge \neg Attached(Leg1, Leg2)$.

In this case, however, we do not assume a fixed value for predicates not mentioned. Rather, we do not care what their values are—any state in which the goal expression evaluates to *True* is a goal. A goal specification can thus be thought of as a *partial state* specification.

A goal can be considered a list of conditions to be met. Thus, the number of those conditions already met provides an obvious search heuristic. We may even try to solve each condition separately, by assuming that the problem is **decomposable**. Very few problems are perfectly decomposable, but many are **nearly decomposable**, where only a little extra work is required to transform a decomposed solution into a full solution.

Actions in PDDL consist of three components:

1. A *name* for the action, including its parameter list.

2. A *precondition*, which is a conjunction of predicates that expresses when the action can be executed. This expression may use the parameters of the action as variables, rather than being restricted to ground literals.

3. An *effect*: a conjunction of predicates describing the result of executing the action. Positive predicates become *True*, and negative predicates become *False*; predicates not listed do not change their truth values.

For example, in our furniture assembly problem we may have the following action:

- *Action(Insert(A, B)):*
  Preconditions: $Part(A) \wedge Slot(B) \wedge Free(A) \wedge Open(B) \wedge \neg In(A, B)$
  Effect: $In(A, B) \wedge \neg Free(A) \wedge \neg Open(B)$

This action allows us to insert part A into slot B. Its preconditions ensure that A is indeed a part and that B is indeed a slot; furthermore, that A is free (unused), that B is an open slot, and that A is not already in B. Executing the action results in part A being in slot B, part A no longer being free, and part B no longer being open.

Note that A and B are variables, and so the action may be applicable to multiple symbols in a state. For example, in a problem with five free parts and two open slots, the action may be executed in 10 distinct ways, by binding A to one of the five parts and B to one of the two slots. This compact method for specifying an action brings us representational flexibility and structure but requires a matching process for each state to find variable bindings that fulfill the preconditions.

Learning activity

Come up with a PDDL specification for the problem of packaging an item and sending it by mail. Assume that you start with a box, stamps, the item, and some tape, and that opening the box to insert the item would tear the stamps and the tape if they have already been applied. Use as few action definitions as you can.

5.3 Planning with state space search

Given a problem description in PDDL, we can now apply standard search techniques to it. The simplest way to proceed is **forward state-search** or **progressive** planning, where we start with a state and use some search method to attempt to find a goal state that matches our goal expression, using a step cost of 1 for each action. Since we have a finite number of literals, the search space must be finite, and so any complete search algorithm is then also a complete planning algorithm. However, progressive planning can be prohibitively expensive for large search problems. It is very easy to write a planning problem specification in a few lines of PDDL that has an extremely high branching factor.

An alternative is **backward-state search** or **regressive** planning, where we start from the goal condition and employ search to attempt to reach one of the start states. This requires us to compute, given a target state, at least one state from which an action can be executed leading to the target state. While this is slightly more difficult than simply checking whether an action can be executed at a given state, it allows us to only consider **relevant** actions: actions which achieve one of the terms in the target state. This restriction can in practice significantly reduce the branching factor of the search while still leaving it complete.

Even so, without a good heuristic both search methods are not able to scale up to truly large problems. One common heuristic is **relaxation**—making the problem easier, solving it, and then using the resulting distance measures (from each state to the goal) as a heuristic function. Relaxation is commonly achieved either by removing preconditions from all of the actions, or by removing negative literals from the action effects. In both cases the resulting problem is very much simpler to solve, and can often be solved directly.

Another approach, **subgoal independence**, assumes that each literal in the goal conjunction can be achieved independently. We can thus solve for each literal independently of the others, and use the sum of the costs for solving all literals as a heuristic. Unfortunately, although it performs well in practice this approach does not result in an admissible heuristic.

5.4 Partial order planning

An alternative approach to planning is to work on each subproblem (literal in the goal conjunction) independently, and obtain a plan for each. This is known as **partial order planning**, because rather than attempting to find an entire sequence of actions that solve the whole problem, we instead find partial plans that solve each sub-problem, leaving unspecified the order of execution of the actions in the

subproblems. Once we have obtained each sub-plan, we may then **linearise** them to form a final plan.

Consider the following planning problem:

- $Goal(JamToast \wedge Tea)$; $Init(Teabag \wedge Bread \wedge KettleFull \wedge Jam)$.
- $Action(BoilWater)$:
  Precondition: KettleFull
  Effect: HotWater
- $Action(MakeTea)$:
  Precondition: $HotWater \wedge Teabag$
  Effect: Tea
- $Action(ToastBread)$:
  Precondition: Bread
  Effect: Toast
- $Action(MakeJamToast)$:
  Precondition: $Toast \wedge Jam$
  Effect: JamToast

We can consider the two goals, *JamToast* and *Tea*, separately, and produce the following sub-plans:

- $Init \rightarrow BoilWater \rightarrow MakeTea \rightarrow Tea$.
- $Init \rightarrow ToastBread \rightarrow MakeJamToast \rightarrow JamToast$.

In order to produce a **total plan**, we must merge the two sub-plans to form a single plan to solve the whole problem. Since in this case the sub-plans are completely independent, any ordering of the sub-plan components that preserves their own internal ordering will do. In some real-world problems, it may be a good idea to only linearise when necessary, in order to keep as many possible solution paths open as possible, in case (for example) an action fails.

Note that in more complex cases where the sub-plans affect each other, a more complex linearisation process is required. In the most complicated setting, the sub-plans are antagonistic: executing one undoes one of the others. Such cases require more sophisticated plan repair mechanisms in order to produce a total plan.

Learning activity

Come up with a real-life example of a system where the goals result in sub-plans that are mutually antagonistic. Suggest a method for resolving the resulting conflict.

5.5 Exercises

1. Consider applying state-space search to the example planning problem in the previous section. Write out the sequence of states visited by a breadth-first progressive planner trying to solve it.
2. Do the same for a regressive planner. Which visits fewer states?

5.6 Learning outcomes

By the end of this chapter, and the Essential reading and activities, you should be able to:

- Write a PDDL specification for a given problem.
- Describe how a search procedure could be used to solve a problem described by a PDDL specification.
- Merge two partial plans into a total plan.

Chapter 6

Natural language processing

Essential reading

Russell, S. and P. Norvig *Artificial Intelligence: A Modern Approach*. (Upper Saddle River, NJ: Prentice Hall, c2010) third edition [ISBN 9780132071482 (pbk); 9780136042594 (hbk)]. Chapter 23. Describes grammar-based approaches to natural language processing.

Further reading

Russell, S. and P. Norvig *Artificial Intelligence: A Modern Approach*. (Upper Saddle River, NJ: Prentice Hall, c2010) third edition [ISBN 9780132071482 (pbk); 9780136042594 (hbk)]. Chapter 22. Considers probabilistic approaches to natural language processing.

6.1 Introduction

One of the most important aspects of human intelligence is language—it is how the vast majority of human knowledge is obtained and transmitted, and it is an area of intelligence in which human capabilities are vastly beyond that of animals.

If we are to build intelligence machines that must interact with humans, or that must obtain knowledge from sources originally prepared for humans, it is crucial that they are able to understand human language.

However, **natural languages** (such as English) are much harder to deal with than **formal languages** (such as logic), which are clearly defined. Sentences and sets of sentences in natural languages tend to be ambiguous (have more than one meaning), show complex structure, make use of idiom, and require significant background knowledge to understand. These difficulties make natural language processing a difficult but important focus of AI.

6.2 The stages of natural language processing

If we are to understand spoken natural language, we must go through the following steps:

1. **Perception**, where a speech pattern (or optical pattern, in the case of optical text processing) is obtained by a sensor, and must be converted to a symbolic representation. This conversion from a continuous (likely noisy) signal to a discrete symbolic representation is made difficult by the many complexities of human speech—accents, vocal differences, speech patterns, etc.

2. **Syntactic analysis** or **parsing**, where a textual representation of a sentence is broken into its constituent parts of speech, usually in the form of a **parse tree**.
3. **Semantic analysis**, where a parse tree is transformed into a set of (usually logical) statements about the world.
4. **Disambiguation**, where sentences that have many possible interpretations are resolved to a single meaning.
5. **Incorporation**, where the information extracted from the previous steps is incorporated into the agent's representation of the world.

In practice, most systems do not perform all of these stages. For example, many commercially available speech recognition packages just perform the perception stage, whereas most natural language understanding systems start from the second stage and receive their input as text. Further, while most of these problems are considered independently, information from one stage may make another's task easier (for example, a speaker's emphasis on a particular word or body language may make disambiguation easier).

Learning activity

Imagine you wish to learn a new language. How would you go about it? Which of the above stages would you need to learn, and which are already present?

In this subject guide, we do not consider the perception stage, and assume that the incorporation stage consists of simply adding logical expressions to a knowledge base. The task of stages 2 to 4 is to go from textual sentences (presumably obtained from a perception system) to logical sentences (which can presumably be added to a knowledge base).

6.3 Syntactic analysis

Although natural languages do not generally have a formal definition, in order to write programs that understand them we usually start by building a formal definition that we use to perform syntactic analysis on given sentences.

6.3.1 Generative grammars

The formal definitions we use to model language take the form of **generative grammars**, which are sets of rules for how to generate valid sentences.

Generative grammars are made up of lists of **rewrite rules**. Each rule has a left-hand and right-hand side, each of which can be made up of some combination of **terminal symbols** (conventionally written in lowercase) and **non-terminal symbols** (conventionally written in uppercase). Terminal symbols are literals—they cannot be expanded further, while non-terminal symbols must be expanded (until the sentence reaches only terminals). Each rule specifies a legal replacement of a pattern (on the left) with a new pattern on the right, for example:

$$AB \rightarrow xB$$

means that wherever the symbols A and B occur next to each other, they may be replaced by the pattern xB. This would allow, for example, the expression $yABz$ to be rewritten as $yxBz$.

Given a grammar G and a starting pattern, we can repeatedly apply rewrite rules until we reach a string consisting of only terminal symbols. Such a string is said to be *generated by* G, or *in the language of* G, where the language of G is the set of all strings that can be generated by G.

Grammars differ in their power and also in the tractability of deciding whether or not a given sentence has been formed by them. Four important classes of generative grammars are, in descending order of expressiveness:

- **Recursively enumerable** grammars, which use rules that are not restricted on either side.
- **Context-sensitive grammars**, which restrict the right-hand side (replacement pattern) to have at least as many symbols as the left-hand side (search pattern). This allows us to specify the **context** of a symbol when writing a rewrite rule for it. The example rewrite rule given above is a context-sensitive rewrite rule, because it restricts the replacement of the non-terminal A by the terminal x to those cases where A is followed by B.
- **Context-free grammars**, which restrict the left-hand side (search pattern) to a single non-terminal symbol. We call this grammar context-free because each rule allows the replacement of a symbol **no matter what surrounds it**. For example, the context-free rule $A \rightarrow CxD$ specifies that the non-terminal symbol A is replaced by the string CxD; this can occur without regard to the string before or after A.
- **Regular grammars**, which restrict the left-hand side (search pattern) to a single nonterminal, and the right-hand side (replacement pattern) to a single terminal symbol, optionally followed by a single symbol.

All of these grammars are important constructs in computational complexity theory, the study of how complex individual problems are and of which can and cannot be solved by machines. For example, the regular languages are equivalent to finite state automata, while the recursively enumerable languages are equivalent to Turing machines. In fact, these grammars appear all through Computer Science; most programming languages, for example, are specified using a type of context-free grammar known as **Backus-Naur Form**, or BNF.

The most popular type of grammars used in natural language processing are context-free grammars, because they are powerful and relatively efficient to parse, although it is now widely accepted that natural languages are probably not context-free. Figure 6.1 shows the rewrite rules and grammar for an example context-free grammar ε_0 given by Russell and Norvig that partially models English.

Note that the grammar uses the | sign as an "or" symbol: when the right-hand side contains $A \mid B$ we may choose to replace the left-hand side with either A or B (but not both). We will use this grammar for the remainder of this chapter. Also note that the grammar is separated into a **lexicon**, which describes the set of terminal symbols available in the language, and a set of grammar rules. Specifying the lexicon separately allows us to change it easily if we need to add new words to the language.

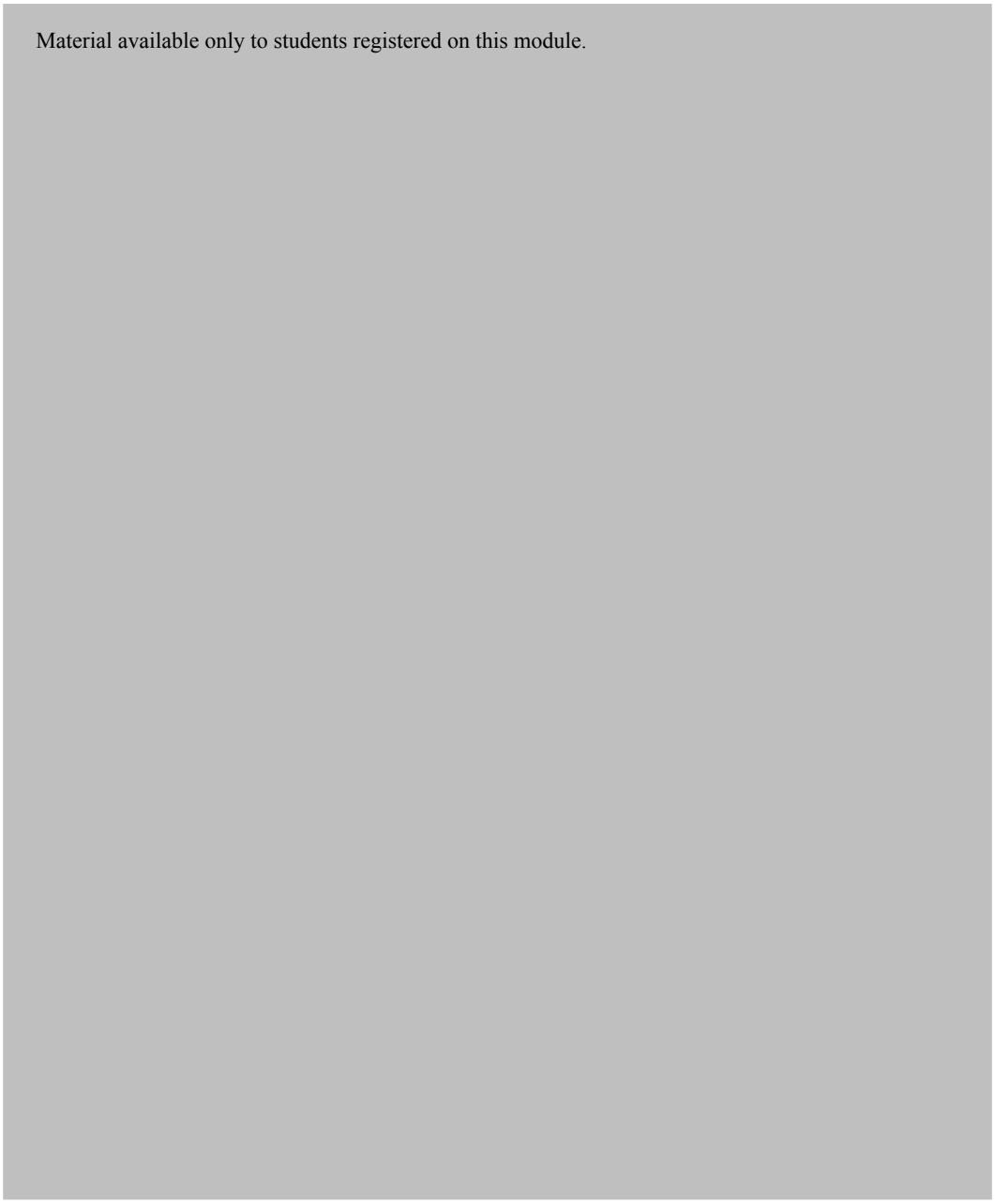


Figure 6.1: ε_0, a formal grammar for a small fraction of English.
Adapted from Russell and Norvig, Chapter 23, page 891. (RUSSELL, STUART; NORVIG, PETER, *ARTIFICIAL INTELLIGENCE: A MODERN APPROACH*, 3rd edition, ©2010. Electronically reproduced by permission of Pearson Education, Inc., Upper Saddle River, New Jersey.)

6.3.2 Parsing

Given a sentence string and a context-free grammar, the program should be able to find a generative sequence of rules that starts with a root symbol (in this case *S*) and obtains a sequence of rewrite rules that obtain the original sentence. This task is called **parsing**. A common output form of parsing is a **parse tree**. Given the example sentence:

I feel a breeze and I smell a wumpus,

we should be able to obtain the parse tree shown in Figure 6.2. The parse tree contains both a label for every word and the structure of the rewrite rules used to generate the sentence. For example, it labels the word *breeze* as a *Noun*, but also as part of a *Verb Phrase*.

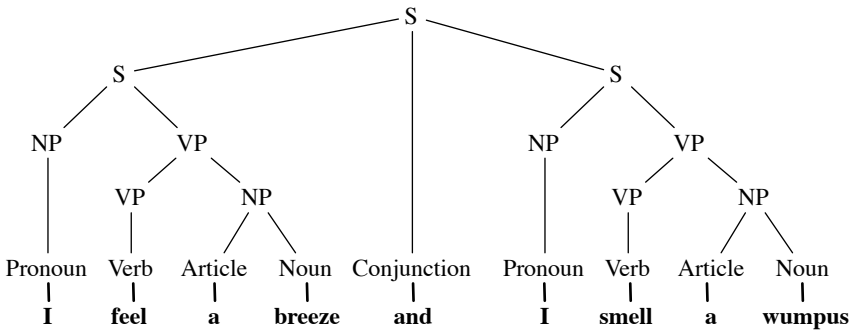


Figure 6.2: The parse tree for *I feel a breeze and I smell a wumpus* using grammar ε_0.

Learning activity

Write down a parse (a sequence of grammar rules starting with S that generates the target sentence) from the parse tree in Figure 6.2.

We can view parsing as a search for a parse tree—in fact, if we consider the left-hand side of a rewrite rule a precondition, and the right-hand side an effect, we can see that the parsing problem is very similar to planning (as discussed in the last section). Thus, we can either perform a **top down** search, starting with S and attempting to generate our target sentence, or a **bottom up** search, starting with our sentence and attempting to reduce it to S, analogously to progressive and regressive planning. More efficient parsers exist, especially for subsets of context-free grammars, but they are beyond the scope of this subject guide.

Learning activity

Construct a parse tree for 'the wumpus to the east stinks'.

6.4 Semantic analysis

The task of semantic analysis is to transform a sentence during and after parsing into a set of sentences in whatever representation language the agent is using (we will assume first-order logic), to be added to its knowledge base. For example, we would like to be able to transform the natural language sentence:

The wumpus is dead,

and obtain the logical sentence:

$$Dead(Wumpus),$$

where *Dead* is a predicate and *Wumpus* is a symbol in our knowledge base.

The parse tree structure obtained by parsing contains a great deal of information about the structure of a sentence; furthermore, natural languages have the property of **compositional semantics**, which means that the semantics of sentences are built out of the semantics of their constituent parts. Therefore, a natural way to obtain semantic information is to build it up during parsing.

We use an **augmented grammar** for this task, where each parse rule is augmented with a rule for building a logical sentence.

In order to do so, we must make use of **λ-notation**, a formalism for writing parameterised functions. We write a function with its parameters explicitly listed and prefixed by λ, for example:

$$f = (\lambda a\ \lambda b\ Sees(b, a)).$$

This notation indicates a function that takes arguments a and b, and everything else inside the expression is a logical symbol and can be treated as a constant. We can then apply an argument to the above expression, so $g = f(Wumpus)$ becomes:

$$g = (\lambda b\ Sees(b, Wumpus)).$$

Naturally we can apply as many arguments as we like. However, the ability to only partially specify the arguments to the function makes λ-notation useful for augmented grammars (the arguments are always specified in the order they are given in the λ-expression).

Figure 6.3 shows a very small example grammar augmented with semantic information, using λ-calculus, and the resulting parse-tree for an example sentence. Each side of each production rule contains logical expressions in brackets.

A further complication in semantic analysis that the semantics of the current sentence may rely on the current **context**. For example, upon receiving the sentence *I smelled a wumpus here*, the agent must attach a suitable referent for *I* and also for *here*. This is known as **pragmatic interpretation**.

Learning activity

Suggest a few rules for pragmatic interpretation of pronouns.

6.5 Ambiguity and disambiguation

All of the example sentences we have considered so far have exactly one possible meaning. However, natural language sentences are very often **ambiguous**, having more than one possible meaning. Potential sources of ambiguity are:

$$
\begin{array}{rl}
S(rel(obj)) \rightarrow & NP(obj)\ VP(rel) \\
VP(rel(obj)) \rightarrow & Verb(rel)\ NP(obj) \\
NP(obj) \rightarrow & Name(obj) \\
NP(obj) \rightarrow & Article\ Noun(obj) \\
 & \\
Name(Mary) \rightarrow & \textbf{Mary} \\
Noun(Wumpus) \rightarrow & \textbf{Wumpus} \\
Verb(\lambda a\ \lambda b\ Smells(b, a)) \rightarrow & \textbf{smells} \\
Article \rightarrow & \textbf{the}
\end{array}
$$

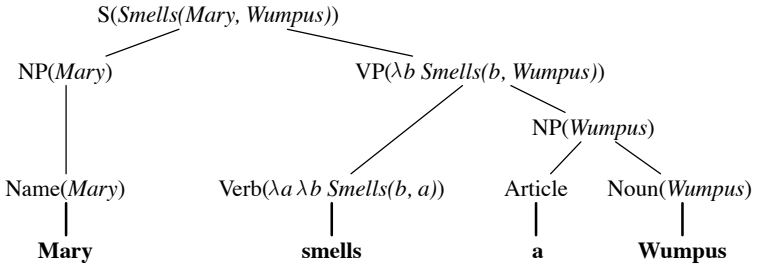


Figure 6.3: A very small example grammar augmented with semantic information, and the resulting augmented parse tree for the sentence *Mary smells a Wumpus*.

- **Lexical ambiguity**, where a single word can have more than one meaning. A common example of this is the word *run*, which is most commonly a verb (meaning the physical act of running (*I like to run*), or the running of a business (*I run a company*), or the running of a program (*please run the spell checker*)) but can also be a noun (*he went for a run*, or *show me the output from your program's third run*).
- **Syntactic ambiguity**, where there is more than one valid parse tree for a single sentence. For example, the phrase *I cut the apple with the knife* could be parsed as *(I cut the apple) (with the knife)* or *(I cut) (the apple with the knife)*. Both are valid parse trees, but since we know that apples do not carry knives, and that cutting is something that is done with a knife, the second is less likely than the first.[1]
- **Semantic ambiguity** most often results from syntactic ambiguity, but can also occur even within a single parse tree.
- Ambiguity between literal and figurative meanings. The most common case of this is **metonymy**, where one object is used to stand for another. For example, the phrase *Wall Street gains record profits* does not imply that a physical street is making any money. Another common form is a **metaphor**, variants of which are common in spoken language. For example, *this mobile phone model is very hot right now* implies the phone is selling well, but says nothing about its temperature.

Thus, the parsing and semantic analysis stages of natural language processing are likely to return **multiple** perfectly valid semantic trees, and we must go through a disambiguation stage to decide which one to use. This is a question of determining

[1] Note, however, that there may be situations where the second is equally likely, or even more likely. Imagine a person holding an apple slicer and faced with two plates. On each plate is an apple, and on one plate is a butter knife. *The apple with the knife* might disambiguate between which of the apples was cut with the apple slicer.

which of the semantic parse trees is more **likely**, and so we can apply the probabilistic reasoning methods we discussed in Chapter 4, along with evidence from our general knowledge of the world, our current model of the world, a model of the language or a speech model.

Learning activity

Suggest one method for determining that a sentence is a metaphor, and determining the meaning of that metaphor. What knowledge do you need?

6.6 Exercises

- Consider a natural language processing system for a drive-through fast-food chain. The system should be able to take people's orders via a speaker-phone and transmit them to the kitchen and billing system. Write down all the reasons why such a system is easier to build than a full natural language processing system.
- Natural languages are generally thought to be context-sensitive, rather than context-free. Why? Give an example.

6.7 Learning outcomes

By the end of your study of this chapter, the Essential reading and activities, you should be able to:

- Determine whether a sentence is in the language of a given grammar and if so produce a parse for that sentence.
- Produce a parse tree for a sentence string, given a grammar.
- Produce an augmented parse tree for a sentence string, given a grammar augmented with semantic information.
- Identify ambiguous sentences, explain why they are ambiguous, and determine and argue for their most likely meaning.

Chapter 7

Machine learning

Essential reading

Russell, S. and P. Norvig *Artificial Intelligence: A Modern Approach*. (Upper Saddle River, NJ: Prentice Hall, c2010) third edition [ISBN 9780132071482 (pbk); 9780136042594 (hbk)]. Chapter 18. Introduces supervised learning.

Russell, S. and P. Norvig *Artificial Intelligence: A Modern Approach*. (Upper Saddle River, NJ: Prentice Hall, c2010) third edition [ISBN 9780132071482 (pbk); 9780136042594 (hbk)]. Chapter 21. Introduces reinforcement learning.

Further reading

Russell, S. and P. Norvig *Artificial Intelligence: A Modern Approach*. (Upper Saddle River, NJ: Prentice Hall, c2010) third edition [ISBN 9780132071482 (pbk); 9780136042594 (hbk)]. Chapters 19 and 20. Describes more sophisticated approaches to supervised learning through the inclusion of prior knowledge and the use of probabilistic methods,

Sutton, R.S. and A.G. Barto *Reinforcement Learning: An Introduction.* (MIT Press, 1998) [ISBN 9780262193986]. The standard introductory textbook on reinforcement learning. An HTML copy is available on the Internet at:
`http://webdocs.cs.ualberta.ca/~sutton/book/the-book.html`

Murphy, K.P. *Machine Learning: A Probabilistic Perspective.* (MIT Press, 2012) [ISBN 9780262018029]. A good introductory textbook on machine learning, with an emphasis on probabilistic methods.

7.1 Introduction

Machine learning is a subfield of AI concerned with creating systems that are able to improve their performance on some given task with experience.

A great deal of what we consider to be intelligence in humans is learned, through many years of experience in the world or in formal settings (such as at school). Learning is part of what makes humans so versatile, and so it is hoped that systems that are able to learn what to do, rather than having to be programmed with what to do, might show similar versatility.

Machine learning can be split into three primary categories, depending on the type of feedback that the system receives:

- **Supervised learning**, where the system learns by **examples**. Such a system is typically presented with a set of example input/label pairs and must use them to

learn to present the correct label for new inputs. In this setting, during learning the agent is always given the correct answer alongside the examples, and is typically trying to learn a **concept**. For example, we might show the system pictures of cars (labelled *car*), and pictures of other stuff (labelled *not car*), and then see if it is able to identify new pictures of cars. This is known as supervised learning because it requires a 'teacher' who labels the examples for the agent.

- **Unsupervised learning**, where the system is simply presented with input data. The data is not labelled; rather than learning to label it, the system instead must attempt to learn useful properties of the data as a whole. An unsupervised learning system might, for example, learn that there are distinct classes of inputs in the input patterns it gets, without having been told. This information is often used later on to improve the system's performance. For example, a system might be given a series of spoken words pulled at random from some audio book podcasts as input. The system may learn that there are three distinct voices in the input data. This information might prove very useful if the system is later asked to learn which spoken sentence fragments come from which book.
- **Reinforcement learning**, where the system must interact with an environment, and is given a **reward** for each action it takes. The goal of the agent is then to learn to maximise the sum of the rewards it receives over time. Unlike supervised learning, the agent is never told what the 'correct choice' is; instead, after a long sequence of selection, it is given an overall 'reward' that indicates how good the entire sequence is (although it might be given intermediate hints). The 'answer' is a **control policy**, not a concept. For example, a reinforcement learning agent might be used to learn to play chess: it may be given small negative rewards for losing pieces, small positive rewards for taking its opponent's pieces, a large positive reward for winning the game and conversely a large negative reward for losing it. Using just this information, the agent should learn to play a good game of chess.

In this chapter we will consider simple example learning algorithms for supervised learning and reinforcement learning. Machine learning in general is a vast and complex enterprise, and so the interested reader is referred to the Further reading list for further coverage.

Learning activity

Think of an example of two of the above types of learning from your own life. For each example, write down a description of the learning task you had to perform, how long it took you to accomplish it, and what background knowledge you needed to have to be able to learn it.

7.2 Supervised learning

In supervised learning we are concerned with learning a function from some set of input variables to some set of output variables. Supervised learning is typically thought of as splitting into two broad types of learning:

- **Classification**, where the outputs that the system is trying to learn are a set of discrete **classes**. For example, given labelled pictures of various modes of

transport the system might be tasked with learning to classify new pictures as planes, trains or automobiles.

- **Regression**, where the outputs that the system is trying to learn are continuous. For example, given various meteorological measurements the system might be tasked with learning to predict tomorrow's temperature.

In this section we will consider classification problems only.

One of the major challenges of classification is **generalisation**. Given a labelled set of examples, the task of the learner is to be able to generalise to input examples it has not yet seen. If it cannot, then learning is pointless. For example, consider a learner that classifies images of written digits into letters of the alphabet. We might wish to use such a system to extract text from images of documents, but the system would be completely useless if it is only correct for the exact example images that it has already seen.

Because of this, classifiers are usually tested on a different set of labelled examples to the ones they are trained on. Thus, the **training set** of examples is used for learning, and the **test set** of data is used to measure how well the learning system is able to generalise to unseen examples.

7.2.1 Decision trees

Decision trees are a relatively simple, popular method for classification when the input consists of variables (termed **attributes**) with a finite number of possible values (termed **attribute values**). Each node in a decision tree is either a **decision** or a **test**.

Decisions are always leaf nodes labelled by one of the classes the decision tree can output, and reaching one for a particular input indicates that the tree returns that class for that input. Internal nodes are always tests, consisting of the name of one of the attributes, along with an outgoing edge for each of its possible attribute values. Figure 7.1 shows a simple decision tree for determining whether a run in cricket was a boundary, and if so whether it scored 4 or 6. Decision nodes are shown in bold with rounded edges.

When considering a particular input, execution starts at the root node. For each test, we follow the edge corresponding to that attribute's value for the given input. For example, given input data *[PassedBoundary = Yes, Bounced = No]* we first test *PassedBoundary*; following its *Yes* edge, we test *Bounced*; following its *No* edge, we can conclude *Boundary, 6 runs*.

Inducing decision trees

A decision tree is simply a representation; now we must consider how to induce a decision tree from a set of training examples. Consider the example data in Figure 7.2, which is data gathered for determining under which circumstances someone will decide to take a taxi after going to the cinema, instead of simply walking home.

We may induce a decision tree as follows. First, we pick an attribute to **split** on, and create a node for it. We will construct a separate tree for each possible attribute value by partitioning the examples into one set for each attribute value, each set

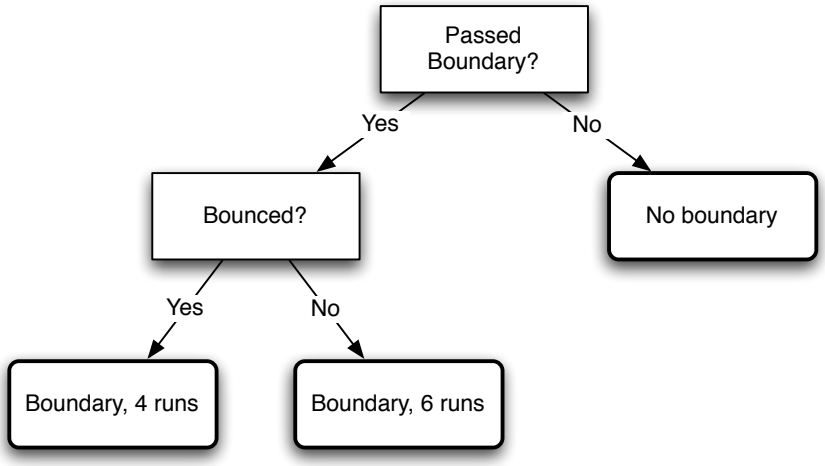


Figure 7.1: A simple decision tree for determining whether a hit ball in cricket scores 4, 6, or is not a boundary.

#	Raining	Cold	Late	Nearby	Umbrella	Warmly Dressed	*Taxi?*
1	Yes	No	Yes	Yes	Yes	No	No
2	No	No	Yes	Yes	No	No	No
3	No	Yes	Yes	Yes	No	No	Yes
4	Yes	No	Yes	No	Yes	Yes	No
5	Yes	No	No	No	No	Yes	Yes
6	No	No	No	Yes	Yes	No	No
7	No	Yes	Yes	No	Yes	Yes	Yes
8	No	Yes	No	No	No	Yes	Yes
9	Yes	No	No	No	No	No	Yes
10	No	Yes	Yes	No	Yes	No	Yes
11	Yes	No	No	No	Yes	Yes	No
12	No	Yes	No	Yes	No	Yes	No
13	No	Yes	Yes	Yes	Yes	Yes	No
14	Yes	No	Yes	Yes	No	No	Yes

Figure 7.2: Example data for the learning to determine whether to take a taxi or walk home.

containing only examples where the attribute we use to split has that particular value. For example, given the data from Figure 7.2, we choose to split on attribute *Raining*, resulting in a set for *Raining=Yes* (examples 1, 4, 5, 9, 11 and 14) and one for *Raining=No* (examples 2, 3, 6, 7, 8, 10, 12 and 13).

We then consider each example set in isolation. If all of the examples in a given example set have the same output label, we create a decision node with that label, add an edge from the split node to the decision node using the attribute value used to obtain the example set, and we can throw away that example set. If, on the other hand, the examples do not all have the same output label, then we recursively create a new decision tree for the example set instead of a decision node.

For example, for the attribute set *Raining=Yes* we have examples 1, 4, 5, 9, 11 and 14, which do not have all the same output values. Now considering only those

examples we start again. Splitting on attribute *Umbrella*, we see that *Umbrella=Yes* leaves us with examples 1, 4 and 11. All of these have output label *Taxi=No*, so we may create decision node *Walk Home* and be done with that attribute setting. For attribute setting *Umbrella=No*, we are left with examples 5, 9 and 14. Again, these three examples all have output label *Taxi=Yes*, and so we can create a decision node *Take Taxi* for that attribute value and stop. Applying the same method for the case *Raining=No* results in the decision tree shown in Figure 7.3.

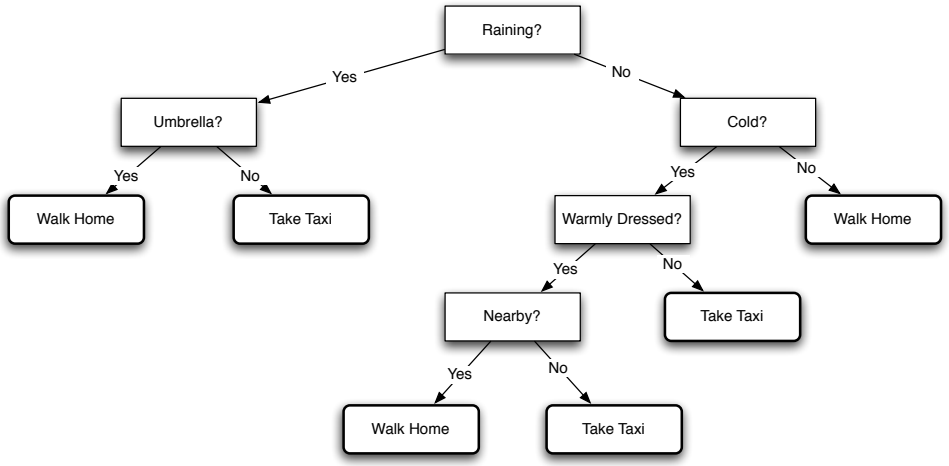


Figure 7.3: A decision tree induced from the data in Figure 7.2.

The decision tree in Figure 7.3 shows some interesting properties. First, we might notice that it decides to walk home in the rain with an umbrella even if it is cold and the decision maker is poorly dressed. This is accounted for by the fact that the data contains no examples where *Raining=Yes* and *Cold=Yes*. In general, a learned decision procedure can only be as good as the data it is given.

Note also that we could have produced a decision tree where each path through the tree accounted for exactly one example, returning *No* everywhere no data was seen. Thus, we would have a tree of depth 6 (because there are six attributes) that perfectly fitted the data we have been given, but would generalise badly to new data. This is known as **overfitting**.

Learning activity

Consider a data set with n examples, each made up of m attributes with two possible attributes each. What is the maximum depth of tree that can be obtained from such a data set? What is the minimum?

Selecting attributes to split on

So far we have not yet specified how to pick which attribute to split on. Ideally, we would prefer to split on attributes that split the data into large groups of examples with the same label. One useful measure of attribute usefulness is **information gain**.

To describe information gain, we require a few new concepts. The **information** I contained in a set of examples can be expressed as a measure of the fraction of positive and negative examples. Let f_{A+} be the fraction of the examples in example set A that are positive, and f_{A-} the proportion that are negative. Then the **information** contained in A can be expressed as:

$$I(A) = -f_{A+} \log_2 f_{A+} - f_{A-} \log_2 f_{A-}$$

Information measures the amount of information required in a set of examples for predicting the label. For example, if A is the whole set of examples, then we see that there are 14 examples, seven of which are positive and seven of which are negative, so $f_{A+} = \frac{7}{14}$, and similarly $f_{A-} = \frac{7}{14}$. Then

$$I(A) = -\frac{7}{14} \log_2 \frac{7}{14} - \frac{7}{14} \log_2 \frac{7}{14} = -\log_2 \frac{1}{2} = 1.$$

Notice that if all of the examples have the same label then the information is zero: we need zero information to determine the labels of the data.

We would like to pick the attribute that results in example sets with the least information. If we have attributed B with v potential values, then we can express the *information gain* from choosing that attribute as:

$$Gain(B) = I(A) - \sum_{i=1}^{v} f_{B_i} I(B_i)$$

where B_i is the set of examples where attribute B takes value i, and f_{B_i} is the fraction of examples in A that are also in B_i. We can thus consider the information gained for splitting on each attribute, and choose the attribute with the largest gain.

Learning activity

We have seen above that information is zero when all the attributes have the same label. Given n examples, what allocation of labels to them will maximise information?

7.3 Reinforcement learning

Reinforcement learning studies the problem of learning to maximise a numerical reward signal over time in a given environment. As a reinforcement learning agent interacts with its environment, it receives a reward (or sometimes cost) signal for each action taken. The agent's goal is then to learn to act so as to maximise the cumulative reward it receives in the future. Figure 7.4 shows a simple reinforcement learning environment where the agent must navigate through a maze to find a goal.

7.3.1 Markov decision processes

In the simplest setting, a reinforcement learning agent's environment consists of a small number of distinct states, and is modelled as a **Markov Decision Process (MDP)** described by a tuple:

$$M = \langle S, A, P, C \rangle,$$

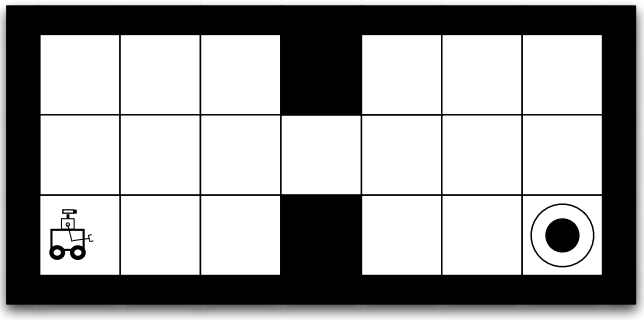


Figure 7.4: A simple reinforcement learning problem.

where S is the set of **states** that the agent may find itself in; A is a set of **actions** that the agent may execute; $P(s'|s,a)$ is a transition function that describes the probability of moving to a state $s' \in S$ from another state ($s \in S$) given an action ($a \in A$); and $C(s,a)$ is a **reward function** that returns a scalar reward signal r for executing action a to state s. Often an MDP is **episodic**, which means that when the agent enters some states, it gains a terminal reward and the entire problem is reset. This allows us to model tasks that the agent can perform repeatedly in order to improve its performance.

For example, in Figure 7.4, the agent can be in one of 19 states (each corresponding to a square in the grid); it has four actions (up, down, left and right); its transition function expresses the fact that an action moves the agent in the direction of the action unless it is blocked by a wall; and its return function is -1 for every step, unless it reaches the goal, in which case it is 100. Once the agent hits the goal the environment is reset to the beginning.

An important property of MDPs is the **Markov property**, which is that for any state s, all the information the agent needs to make a decision about what to do next is contained in s. This means, for example, that we do not need to worry about the agent's history.

The agent attempts to learn a **policy** (mapping from states to actions) π. A policy tells the agent what to do where. Rather than attempting to minimise its immediate reward, the agent wishes to find a policy that maximises its cumulative reward (termed **return**):

$$R_\pi = \sum_{i=0}^{\infty} \gamma^i r_{i+1},$$

where $0 < \gamma \leq 1$ is a **discount rate** that expresses the extent to which the agent prefers immediate reward over delayed reward, and r_i is the reward received at time i.

Figure 7.5 shows an optimal policy for the problem in Figure 7.4. Because the environment is Markov, an optimal policy mapping states to actions cannot be improved by including any other information about the environment.

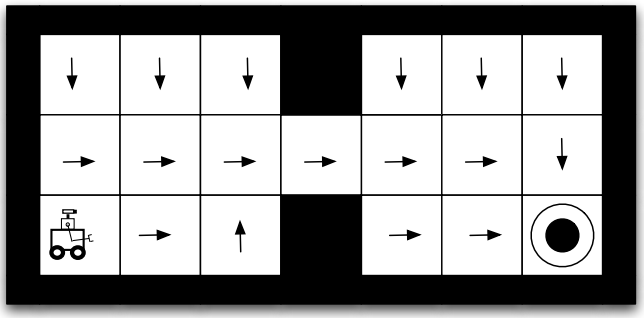


Figure 7.5: An optimal policy for the reinforcement learning problem in Figure 7.4.

Learning activity

Calculate the return for each open square in Figure 7.5, assuming a discount factor of $\gamma = 0.95$. What do you notice about the resulting values as compared to the one-step costs?

This is what makes reinforcement learning hard: the agent wishes to build a policy π to obtain a maximum R_π, but is only given short-term rewards (r_i) at each step.

Note that reinforcement learning has similar objectives to planning: if we knew the transition and reward functions of the problem we could apply a planning algorithm instead of having to learn. Reinforcement learning is generally applied to problems in which we do **not** know the transition and reward information, and where trying to learn it directly (in order to apply planning) is too hard. Thus, a reinforcement learning agent in the problem shown in Figure 7.4 starts out not knowing the structure of the environment; all it is able to do is perceive discrete states, take actions, and observe their effect.

7.3.2 Value functions

Since we wish to optimise a measure of return (rather than reward), it makes sense for us to attempt to model return. This is usually accomplished through the use of a **value function**, which maps states and actions to return. For example,

$$Q_\pi(s, a) \rightarrow R_\pi.$$

This type of value function is known as a **state-action value function** because it stores the return for a state-action pair, assuming the agent follows policy π. We have a small, finite number of states and actions, and so we may simply store this function in a lookup table, with one element for each individual state.

Since we do not have access to this function, we must learn it. **Reinforcement learning is often primarily concerned with learning a value function.** The most useful property of a value function is that it should have a zero **Bellman error**:

$$Q_\pi(s, a) - (r + \gamma Q_\pi(s', a')) = 0,$$

when the agent (following policy π) was in state s, executed action a, then found itself in state s', where it (following π again) executed action s'.

This is equivalent to stating that there should be no difference between the value of a state-action pair and the reward it receives plus the value of the state-action pair that follows it (discounted by γ). This is a consequence of our definition of return, and is known as a **temporal difference error** because it describes the error between the value function at two successive time steps.

Learning activity

Verify that the return values you computed for Figure 7.5 in the previous exercise have zero Bellman error for the optimal policy.

7.3.3 Temporal difference learning

We may use this property to derive a learning algorithm, whereby when the agent takes an action we update its value function to reduce its temporal difference error. Thus, for a given (s, a, r, s', a') tuple from the environment, we can perform the following update:

$$Q_\pi(s,a) = Q_\pi(s,a) + \alpha[(r + \gamma Q_\pi(s',a')) - Q_\pi(s,a)],$$

for some **learning rate** $0 \leq \alpha \leq 1$. Note that we do not include the $\gamma Q_\pi(s', a')$ term for actions that end an episode. Since this family of methods updates the value function to minimise temporal difference error, they are known as **temporal difference methods**. This particular algorithm is known as **Sarsa**.

Of course, learning a value function by itself does not help us to improve our policy. Since we usually start with a random policy (where all actions are equally likely), we will not obtain a very high return; we require a method for updating our policy so that it gains us a higher return. A simple method for doing so is **policy iteration**, whereby we generate a new policy from a value function:

$$\forall s, \pi(s) = \arg\max_a Q(s,a),$$

and so we define the action taken at a state as the action that maximises the value of the state-action pair for that state. Under some fairly standard conditions (which we will not consider here), any interleaving of temporal-difference learning and policy iteration is guaranteed to converge to within a factor of the optimal value function (and hence the optimal policy).

7.4 Exercises

1. For the data in Figure 7.2, write a program that constructs a decision tree using information gain to select splits. What is the depth of the tree? Modify your program to select splits at random, and generate 100 trees. Record the maximum, minimum and average depths of those trees. How do they compare with the tree generated using information gain?

2. Write a program to simulate the problem given in Figure 7.4, and write a reinforcement learning agent program to solve it, starting with a random policy, a discount factor of $\gamma = 0.95$, and a state-action value function initialized to 100 for all states and actions. Perform policy iteration after every step (i.e., define the policy at a state as the action with the highest current Q value, breaking ties randomly). Plot a learning curve that shows the number of steps (averaged of 100 runs) taken to solve the problem against the number of the episodes the agent has experienced. Try α values of 0.1, 0.01 and 0.001 and plot the resulting learning curves. What effect does α have?

7.5 Learning outcomes

By the end of your study of this chapter, the Essential reading and activities, you should be able to:

- Determine whether supervised learning, unsupervised learning, or reinforcement learning is appropriate for a given problem solution.
- For an appropriate classification problem, implement a decision tree using information gain to determine which attributes to split on.
- Describe a reinforcement learning problem as an MDP, given a textual problem description.
- Implement Sarsa to solve a reinforcement learning problem.

Chapter 8

Philosophy

Essential reading

Russell, S. and P. Norvig *Artificial Intelligence: A Modern Approach*. (Upper Saddle River, NJ: Prentice Hall, c2010) third edition [ISBN 9780132071482 (pbk); 9780136042594 (hbk)]. Chapter 26.

Further reading

Haugeland, J. (ed.) *Mind Design II: Philosophy, Psychology, and Artificial Intelligence.* (MIT Press, 2000) second edition [ISBN 9780262581530]. A compilation of important essays written by leading researchers and philosophers on the fundamental ideas behind their approaches to achieving AI.

Lycan, W. and J. Prinz (eds) *Mind and Cognition: An Anthology.* (Wiley-Blackwell, 2008) third edition [ISBN 9781405157858]. A collection of important essays by leading philosophers of mind.

8.1 Introduction

The nature of thought and the mind were considered philosophical questions long before the invention of the computer and the field of AI.

Philosophical claims about AI can be broken into two distinct areas, **Weak AI** claims and **Strong AI** claims.

8.2 Weak AI: can machines act intelligently?

The Weak AI claim is the claim that a computer program can be built that **acts intelligently**; in other words, that is able to solve problems or handle tasks that would typically be considered to require intelligence.

Since this argument is essentially about **behaviour**, it avoids the question of whether or not a machine can 'think' or be considered conscious.

8.2.1 The Turing test

In a seminal paper in 1950, Alan Turing introduced a behavioural test for intelligence now widely known as the **Turing test**.

In the Turing test, an interrogator has a five minute conversation (via textual questions and answers) with two parties, one of which is a computer, and one of which is a human. The interrogator must determine which is the human, and which is the machine; if they fail 30 per cent of the time the computer is said to have passed the test.

Turing claimed that any machine that passes such a test should be considered intelligent, and that it would one day be possible to build such a system.

The Turing test was a landmark thought experiment because it set out a clear, decisive target around which arguments for and against the possibility of building intelligent machines could be made. Moreover, such a system would need to be able to do a great deal, for example, it would need to:

- be able to understand and generate sentences in English
- possess common sense about the world
- make inferences about the world from its common sense.

However, the Turing test is only one particular form of the Weak AI claim: that an intelligent system could be built that **can communicate as if it were a human**, and the test itself has many drawbacks. The system does not have to interact for more than a short period of time; nor does it need to be able to interact with the world, other than by typing answers. It may be advantageous for the program to lie (if asked 'what colour is your hair?') or pretend to perform worse than it could ('what is the hundred and sixteenth decimal place of π?'). Finally, the system is considered intelligent only in its duplication of human intelligence, which we can argue is not the only form of intelligence.[1]

Turing anticipated several objections to his claim, including:

The argument from disability can be paraphrased as: **but a machine can never do X**, for some X, such as 'write a poem' or 'play a game of chess'. As the preceding example illustrates, in many cases some time after a particular activity is claimed to be impossible a system that can do it is demonstrated. Although there are many important activities that computers have not yet been shown to be able to do, it is important not to confuse **not yet knowing how to create such a system** with **such a system being impossible**.

The mathematical objection argues that, since there are some questions that are provably unanswerable by computers, and that since humans can presumably answer them, a computer can never be as intelligent as a human. This argument relies on the assumption that humans are free from such limitations, which has not been shown to be the case. A related objection, **the argument from informality**, which states that computers are formal (mathematical) systems and human intelligence is not, and therefore cannot be modelled by such systems.

Learning activity

Look up the other objections anticipated by Turing, select one, and write a brief summary of it along with Turing's reply. Do you find the reply convincing?

---

[1] Star Trek's Commander Data, for example, would quickly fail the Turing test, but most people would nevertheless agree that he is intelligent.

8.3 Strong AI: can machines think?

The **Strong AI** claim goes one step further than the Weak AI claim. The Strong AI Claim is that **a machine that behaves like a human has a mind and is capable of actually thinking**, as opposed to simply carrying out a 'simulation' of thinking.

This question has its root in the **mind–body** problem, which asks how the physical states of the body (specifically, the states of neurons in the brain) relate to mental states.

Dualism holds that the body and mind (or soul) are two different things, with the mind or soul existing outside of the physical plan. **Materialism**, on the other hand, holds that only material objects exist, so physical states **are** mental states: **'brains cause minds'**.

If one accepts materialism, then the question comes down to whether or not the functional or physical properties of the brain cause minds. **Biological naturalism** holds that minds are a result of the physical properties of the brain—**neurons cause minds**—in which case a simulation would just be a simulation. By contrast, **functionalism** holds that mental states are caused by the computational properties of the brain—it is their causal role in input and output that matters.

Learning activity

Decide on a mind–body position (dualism, biological naturalism or materialism) and write down an argument for why your position is correct. Make your argument as strong as possible, with as few assumptions as possible.

8.3.1 The Chinese room

The most famous argument against the Strong AI claim is Searle's **Chinese room**. The Chinese room appears to understand Chinese: it accepts pieces of paper with questions in Chinese through a slot, and a few minutes later outputs the answers, again written in Chinese, through another slot.

Inside the room sits a man who only speaks English, with a large stack of books with English rules for translating Chinese symbols (which are just 'squiggles' to the man) to other Chinese symbols. When a new piece of paper enters the room, he follows the rules in the books to produce a new page, and pushes that page through the output slot.

Searle's claim is that, although it appears that the room understands Chinese, it cannot, since none of its components do—the man only understands English, and all the other elements are inanimate objects. If we accept this, and substitute 'man' for 'computer', we see that a computer program that behaves as if it understands Chinese (and could thus pass the Turing test, at least with a Chinese interrogator) does not actually understand it.

There are several counterarguments to Searle's claims, of which we will handle only

one here: the **systems argument**, which says that although none of the individual components of the room understand Chinese, the room as a whole—as a system—does. Analogously, consider each of the neurons in a person's brain: in which neuron is the understanding of English? This boils down to the basic mind–body problem: can a collection of simple things (atoms, neurons) be arranged in such a fashion as to result in a mind, even though none of them individually are one? According to the systems argument, the answer must be yes.

In any case, most AI researchers do not believe that arguments about Strong AI are important: constructing intelligent systems is, in their view, sufficiently interesting, whether or not such systems match our current meaning of the word 'mind'.

Learning activity

Look up the other replies to the Chinese room argument in Russell and Norvig, select one, and briefly summarise it. Of all of the replies, which do you find the most convincing?

8.4 Social and ethical implications of AI

Like any other emerging and potentially transformative technology, AI is having and will continue to have, a significant impact on society. Russell and Norvig list six potential threats to society:

- job losses
- too much (or too little) leisure time
- humans might lose their sense of 'uniqueness'
- potential loss of privacy
- potential loss of accountability
- AI systems might take over the world.

Of these, the two most significant at the moment are the potential for job losses and the potential loss of accountability.

Automated manufacturing has already resulted in a reduction of jobs in many industries, and more can be expected to follow as fielded AI systems become more common. In addition, many professions will be as completely changed by the arrival of AI systems as they have been by the arrival of cheap computers. Although this is balanced to some extent by the creation of new jobs facilitated by such systems, the impact of AI systems would be significant in almost every industry.

Once AI systems enter an industry, they open up new questions of accountability. Consider medical expert systems: these systems today commonly perform complex diagnoses automatically. If they make an error, who is liable? While current systems are mostly considered advisory with a doctor making the final decision, as they become more complex this may cease to be the case.

Most of the above threats (with the exception of the threat to uniqueness and that of AI systems taking over the world) are similar to those posed by any new

technology. The threat of AI systems taking over the world can be best considered, at least for the moment, highly unlikely—except to the extent that we already rely heavily on computers.

Finally, there is one further interesting implication of AI systems. If we were to build an artificially intelligent robot that claimed to be conscious, to what extent should it be considered a person, and to what extent simply as a machine? These questions lead to the possibility of a civil rights movement for robots.

Learning activity

Think of an instance of an automated system which has privacy implications for its users. Can privacy safeguards be put into place to compensate for the potential loss of privacy? Decide whether you think the benefits of the system outweigh the risks or not, and write down an argument for why you think so.

8.5 Exercises

1. Consider a version of the Turing test where a man and a woman are trying to convince an interrogator that they are a man. If the woman always succeeds, is she a man? Explain.
2. Now consider a version where an expert programmer and a novice must convince an interrogator that they are an expert programmer. If the novice can always convince the interrogator, is he or she an expert programmer? Explain.

8.6 Learning outcomes

By the end of your study of this chapter, the Essential reading and activities, you should be able to:

- Describe the Weak and Strong AI claims and explain the difference between them.
- Describe the Turing test.
- Describe the Chinese room thought experiment, and discuss the various replies to it.
- Describe the social and ethical implications (both good and bad) of a given new AI technology.

Appendix A

Sample examination paper

A.1 Rubric

Duration: Two hours, 15 minutes.
Answer FIVE (out of six) questions.
Full marks will be awarded for complete answers to FIVE questions.
There are 100 marks available on this paper.

A.2 Questions

1. Search *(Total: 20 marks)*

   (a) Describe breadth-first, depth-first, and iterative-deepening search. Include their time and space complexities in your description, and whether or not they are complete and optimal. *(10 marks)*

   (b) Imagine that you are constructing a program to find the shortest route via a set of roads to a given destination. Assuming you have the use of all of the information contained in a map of the area, write down a good heuristic for this problem and prove it to be admissible. *(5 marks)*

   (c) Prove that A^* search with an admissible heuristic is optimal. Hint: use a proof by contradiction. Assume that a suboptimal solution is found, and show that this implies that the heuristic must be inadmissible. *(5 marks)*

2. Knowledge representation and reasoning *(Total: 20 marks)*

   (a) Write down a first-order logic knowledge base containing the following knowledge:

   - Everything that can fly is a bird.
   - Only parrots and swallows are birds.
   - Parrots talk.
   - Polly flies.
   - Swallows can glide.
   - Birds that can glide do not flap.
   - Polly flaps.

   Now determine the truth of the question: can Polly talk? Show your working. *(10 marks)*

   (b) Consider the following problem. A test for insecticides in drinking water returns the correct result 99.5 per cent of the time; approximately 0.5 per cent of drinking water is thought to be tainted. If water is tested and the test returns true, is the water most likely to be tainted? *(5 marks)*

(c) Suggest a rule of thumb for deciding which variables to bind first in universal instantiation while attempting to answer a query. Explain why this might be effective, and under which cases it may not be. *(5 marks)*

3. Planning *(Total: 20 marks)*

(a) Write a PDDL problem formulation for the following task. You are required to build a tower five blocks high on top of a table, and are given three red blocks and two blue blocks. You may place a block atop another, but only if they are of a different colour. You may place any coloured block on the table when it is empty. Include an action for removing a block from the top of the stack in case your planner makes a mistake. Write down the solution to the problem. *(10 marks)*

(b) Explain the difference between forward (progressive) and backward (regressive) search as a mechanism for planning. Which one is likely to be more efficient, and why? *(5 marks)*

(c) What is partial order planning? When is it effective, and what are the potential difficulties? *(5 marks)*

4. Natural language processing *(Total: 20 marks)*

(a) Produce a parse tree for the sentence *I see the wumpus and it stinks*, using the ε_0 grammar in Figure 6.1. *(5 marks)*

(b) Produce a semantic parse tree of the same sentence (create predicates as necessary). *(7 marks)*

(c) Transform the parse tree from part b to a set of logical clauses. *(3 marks)*

(d) Given the sentences *John took his car to the mechanic, who got it running. He then drove it back to his home.* What sense of the word *running* is meant in the first sentence? To whom does *he* refer to, in the second sentence? In both cases, explain your answer, and discuss how an automated system might make such inferences, and what knowledge it would need to do so. *(5 marks)*

5. Machine learning *(Total: 20 marks)*

(a) Given the following data set, produce a decision tree for deciding attribute *D*. Use the information gain heuristic for selecting the **first** attribute to split on; thereafter you should select for the shortest tree by inspection. Show all your working. *(10 marks)*

#	**A**	**B**	**C**	**D**
1	Yes	1	No	Yes
2	No	1	Yes	Yes
3	No	1	No	Yes
4	Yes	1	Yes	No
5	No	2	Yes	No
6	Yes	2	Yes	No
7	No	3	Yes	Yes
8	Yes	3	No	Yes
9	No	3	No	Yes
10	Yes	2	Yes	No
11	No	3	Yes	Yes
12	No	2	No	No
13	Yes	3	Yes	No

(b) Consider a robot in a maze with 100 states, with a hidden treasure. The robot must retrieve the treasure (by moving through the maze between adjacent squares, until it is over the treasure and can pick it up) and return to its starting square, whereupon the episode ends. What are the states in this example, and how many of them are there? What are the actions? How many state-action pairs are there? *(5 marks)*

(c) Reward functions in reinforcement learning are usually sparse—for example, when playing draughts, we might give the agent a reward for winning and punish it for losing, but say nothing about what happens before that. What would happen if we added extra rewards into the reward function (for example, a small reward for capturing a piece)? Name one positive and one negative aspect to this approach. *(5 marks)*

6. Philosophy of AI *(Total: 20 marks)*

(a) Define and explain the difference between the *Weak AI claim* and the *Strong AI claim*. *(5 marks)*

(b) If a researcher brought you a system that she claimed proved the *Weak AI claim*, how would you test it? If she further claimed it proved the *Strong AI claim*, how could you test it? Could you test it at all? *(5 marks)*

(c) Explain Searle's Chinese room argument. What is the role of the human in the Chinese room? Could you replace him with something else? *(5 marks)*

(d) If a human has a heart transplant, are they the same person? If they have a 'brain transplant', are they the same person? If we could perfectly duplicate someone's brain and put it in another body, would the two be the same person? Would they behave the same? Explain. *(5 marks)*

A.3 Example solution

1. Search

   (a) Both breadth-first and depth first initially add the start node to a queue. They then repeatedly remove a node from the queue, check whether any of its children are the solution, and if not add them to the queue and repeat until a solution is found. The difference is in what order the nodes are removed from the queue. *(1 mark)*

      i. Breadth-first search opens the node with the smallest depth first. This results in all nodes at depth k being visited before any nodes of depth $k + 1$. *(1 mark)* It has time and space complexity $O(b^{d+1})$ ($O(b^d)$ is also acceptable) in the worst case *(1 mark)*, and is complete when b is finite, but is not optimal. *(1 mark)*

      ii. Depth-first search opens the node with the largest depth first. The search thus explores all of a node's children before it explores its remaining siblings. *(1 mark)* It has time complexity $O(b^m)$ and space complexity $O(bm)$ *(1 mark)*, is not optimal and is only complete if m and b are finite *(1 mark)*.

      iii. Iterative deepening search conducts a depth-first search up to a fixed maximum depth (starting with depth 1) and if that fails to find a solution increases the maximum depth. *(1 mark)* It has time complexity $O(b^d)$ and space complexity $O(bd)$, is complete when b is finite but is not optimal. *(1 mark)*

      Note: b is the branching factor, m is the maximum depth and d is the minimum solution depth of the search tree.

   (b) We could use the straight-line distance between two points as a heuristic. *(2 marks)* This is an admissible heuristic because it always underestimates the distance between two points *(1 mark)*, because the straight line distance is the shortest possible distance between two points and the road distance will thus be at least as long *(2 marks)*.

   (c) (Example) Assume that A^* with an admissible heuristic finds a non-optimal solution A (with cost C_A) to a search problem. Let the optimal solution be B (with cost C_B). Then $C_B < C_A$. *(1 mark)*

      If B has not been found then there is at least one node on its path that was not opened during search; let the first such node be b. Since b is the first unopened node, its parent node must have been opened. The estimated cost for b would have been the path cost to b ($g(b)$) plus the heuristic estimate of the remaining cost from b to a solution ($h(b)$). Since $g(b) + h(b) \leq C_B$ (since $g(b)$ is exact and $h(b)$ is admissible and therefore underestimates the remaining cost), we see that $g(b) + h(b) \leq C_B < C_A$. *(2 marks)*

      Now consider the expansion of the final node in A. Its cost estimate is exact, since it is a solution. However, it was opened instead of b, which was on the queue because its parent had been opened. Thus, it must have been the case that $C_A \leq g(b) + h(b)$, else we would have opened b instead. And so we reach a contradiction. *(2 marks)*

2. Knowledge representation and reasoning

(a) i. $\forall x, CanFly(x) \Rightarrow Bird(x)$ *(all flying things are birds)*

ii. $\forall x, Bird(x) \Rightarrow Parrot(x) \vee Swallow(x)$ *(only parrots and swallows are birds)*

iii. $\forall x, Parrot(x) \Rightarrow CanTalk(x)$ *(parrots talk)*

iv. $CanFly(Polly)$ *(Polly can fly)*

v. $\forall x, Swallow(x) \Rightarrow CanGlide(x)$ *(swallows can glide)*

vi. $\forall x, Bird(x) \wedge CanGlide(x) \Rightarrow \neg Flaps(x)$ *(birds that can glide do not flap)*

vii. $Flaps(Polly)$ *(Polly flaps)*

(5 marks)

Now we ask $CanTalk(Polly)$? First, we apply modus ponens to sentences *i* (instantiated using $Polly$) and *iv*, resulting in:

viii. $Bird(Polly)$.

Then, applying contraposition to sentence *vi*, and instantiating it using $Polly$, we obtain:

ix. $Flaps(Polly) \Rightarrow \neg(Bird(Polly) \wedge CanGlide(Polly))$.

Using sentences *vii* and *viii*, we can conclude:

x. $\neg CanGlide(Polly)$.

Then instantiating sentence *v* using $Polly$ and applying contraposition, we can conclude:

xi. $\neg Swallow(Polly)$.

Since we know $Bird(Polly)$ (sentence *viii*) and $\neg Swallow(Polly)$ (sentence *xi*), we can use rule *ii* to conclude:

xii. $Parrot(Polly)$.

Then instantiating sentence *iii* using $Polly$, we finally derive:

xiii. $CanTalk(Polly)$,

and so we can return *True*.

(5 marks)

(b) We wish to determine whether $P(Taint|T_+) > P(\neg Taint|T_+)$, where $Taint$ represents whether or not the water is tainted and T_+ indicates a positive test. *(1 mark)*

From Bayes' rule, $P(Taint|T_+) = \dfrac{P(T_+|Taint)P(Taint)}{P(T_+)}$.

Similarly $P(\neg Taint|T_+) = \dfrac{P(T_+|\neg Taint)P(\neg Taint)}{P(T_+)}$. *(1 mark)*

$P(T_+)$ occurs in both equations so we may drop it. *(1 mark)*

From our given information, $P(Taint) = 0.0005$, so $P(\neg Taint) = 0.9995$. Similarly, $P(T_+|Taint) = 0.995$ and $P(T_+|\neg Taint) = 0.005$. *(1 mark)*

Thus, we wish to evaluate $0.995 \times 0.0005 > 0.005 \times 0.9995$, which equals $0.0004975 > 0.0049975$. Thus, it is more likely that the water is **not** tainted. *(1 mark)*

(c) (Example) One possible rule is to first consider variables that are present in the target sentence. This would be effective because there may be many variables in the knowledge base but those in the goal sentence are most likely to be relevant. The rule may not be effective when there are many variables used in the path to the solution, and those in the goal sentence only appear very late in the path. *(5 marks for insight and reasonable argument)*

3. Planning
   (a) (Example)
   - $Init$: $Red(A) \wedge Red(B) \wedge Red(C) \wedge Blue(D) \wedge Blue(E) \wedge Top(Table) \wedge Free(A) \wedge Free(B) \wedge Free(C) \wedge Free(D) \wedge Free(E)$
   - $Goal$: $\neg Free(A) \wedge \neg Free(B) \wedge \neg Free(C) \wedge \neg Free(D) \wedge \neg Free(E)$
   - $Action(StackRed(a, b))$
     $PRECOND$: $Free(a) \wedge Top(b) \wedge Red(a) \wedge Blue(b)$
     $EFFECT$: $\neg Free(a) \wedge \neg Top(b) \wedge Top(a) \wedge On(a, b)$
   - $Action(StackBlue(a, b))$
     $PRECOND$: $Free(a) \wedge Top(b) \wedge Blue(a) \wedge Red(b)$
     $EFFECT$: $\neg Free(a) \wedge \neg Top(b) \wedge Top(a) \wedge On(a, b)$
   - $Action(StackTable(a))$
     $PRECOND$: $Free(a) \wedge Top(Table)$
     $EFFECT$: $\neg Free(a) \wedge \neg Top(Table) \wedge Top(a) \wedge On(a, Table)$
   - $Action(Remove(a, b))$
     $PRECOND$: $On(a, b) \wedge Top(a)$
     $EFFECT$: $Free(a) \wedge Top(b) \wedge \neg On(a, b) \wedge \neg Top(a)$

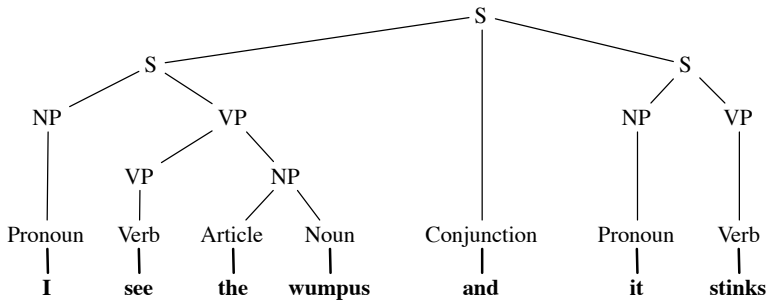
   (8 marks)

   A solution is: $StackTable(A)$, $StackBlue(D, A)$, $StackRed(B, D)$, $StackBlue(E, B)$, $StackRed(C, E)$. *(2 marks)*

   (b) Forward planning starts from the start state in the planning problem and progresses towards the goal. *(1 mark)* By contrast, backward planning starts from the goal specification and attempts to work backwards towards a start state. *(1 mark)* Backward planning is likely to visit fewer states, *(1 mark)* since it visits only states that are relevant (can lead toward the goal). Another way of saying this is that there are likely to be far fewer states on some path to the goal than there are on some path from the start state. *(2 marks)*

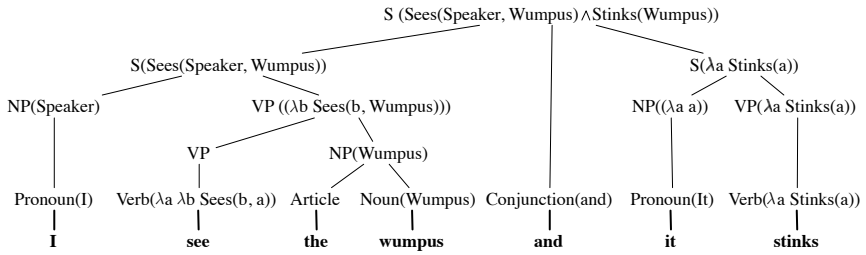
   (c) Partial order planning is a method whereby a plan is built separately for each individual subgoal in the goal conjunction, and then the planner recombines (or linearises) the resulting solutions to obtain an overall plan. *(2 marks)* This may be more efficient than planning for all goals simultaneously because in many cases the subplans are easy to combine because the subproblems are nearly independent. *(2 marks)* However, in some cases it may be very hard to combine the subplans, because they interfere with each other. *(1 mark)*

4. Natural language processing
   (a) Parse tree:



   (5 marks)

(b) Semantic parse tree:



(5 marks for the tree, 2 marks for dealing correctly with the pronouns)

(c) We may add the following sentences to the knowledge base:

- $Sees(Speaker, Wumpus)$
- $Stinks(Wumpus)$

(3 marks)

(d) The word *running* refers to running a machine or a system, not the physical act of running. *(1 mark)* In order to infer this, the system should have information that indicates that running requires legs, and that cars do not have legs. Alternatively, it could have information that indicates that this sense of the word running is often used when speaking of machines, and that a car is a machine. *(1 mark)* The pronoun *he* refers to John, rather than the mechanic. *(1 mark)* In order to infer this, the system should know that a mechanic is unlikely to take someone else's car home, but that John is likely to take his car back home *(1 mark)*. In both cases, the system requires some method for disambiguation, whereby the relative probabilities of both sentences are considered and the most likely chosen. *(1 mark)*

5. Machine learning

(a) We call the data set X. In order to decide on the first split, for each attribute Z we wish to maximise $I(X) - Remainder(Z)$. Since $I(X)$ remains constant, we wish to minimise $Remainder(Z)$.

First we compute $Remainder(A)$.

$$Remainder(A) = \frac{6}{13} I(\frac{2}{6}, \frac{4}{6}) + \frac{7}{13} I(\frac{5}{7}, \frac{2}{7})$$

$$= \frac{6}{13}(-\frac{2}{6}\log_2\frac{2}{6} - \frac{4}{6}\log_2\frac{4}{6}) + \frac{7}{13}(-\frac{5}{7}\log_2\frac{5}{7} - \frac{2}{7}\log_2\frac{2}{7})$$

$$= 0.8886.$$

Next, we compute $Remainder(B)$.

$$Remainder(B) = \frac{4}{13} I(\frac{3}{4}, \frac{1}{4}) + \frac{4}{13} I(0, 1) + \frac{5}{13} I(\frac{4}{5}, \frac{1}{5})$$

$$= \frac{4}{13}(-\frac{3}{4}\log_2\frac{3}{4} - \frac{1}{4}\log_2\frac{1}{4}) + \frac{5}{13}(-\frac{4}{5}\log_2\frac{4}{5} - \frac{1}{5}\log_2\frac{1}{5})$$

$$= 0.5273.$$

Finally, we come to $Remainder(C)$:

$$Remainder(C) = \frac{8}{13} I(\frac{3}{8}, \frac{5}{8}) + \frac{5}{13} I(\frac{4}{5}, \frac{1}{5})$$

$$= \frac{8}{3}(-\frac{3}{8}\log_2\frac{3}{8} - \frac{5}{8}\log_2\frac{5}{8}) + \frac{5}{13}(-\frac{4}{5}\log_2\frac{4}{5} - \frac{1}{5}\log_2\frac{1}{5})$$
$$= 0.8650.$$

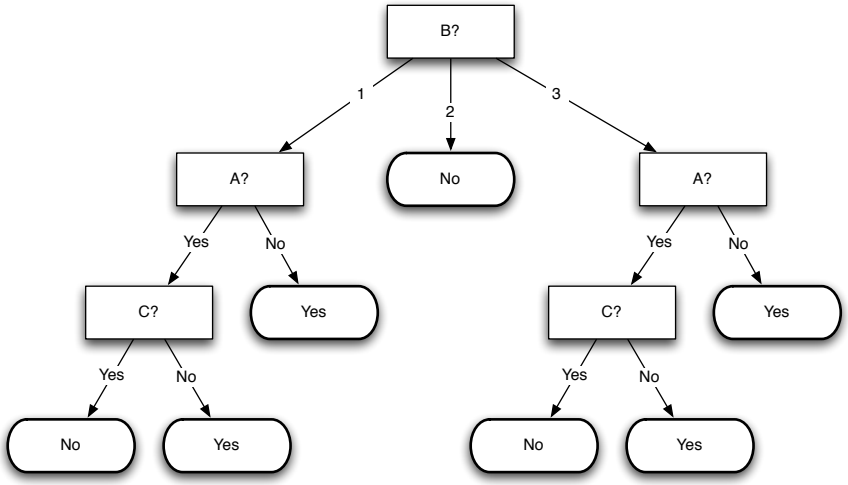
So, we split on attribute B. *(5 marks)*

Inspecting the examples where $B = 1$ (examples 1, 2, 3 and 4), we see that they are all *Yes* except where both A and C are *Yes*. Therefore, we split on A; if it is *No*, we decide *Yes*. If it is yes, we split on C; if C is *No*, we decide *Yes*; if it is *Yes*, we decide *No*. *(2 marks)*

Inspecting the examples where $B = 2$ (examples 5, 6, 10 and 12), we see that all are labelled *No*, so we may simply decide *No*. *(1 mark)*.

Inspecting the examples where $B = 3$ (7, 8, 9, 11 and 13), we see that they are all *Yes* except where A and C are both *Yes*. So, we first split on A; if it is *No*, we decide *Yes*. If it is *Yes*, we split on C; if it is *Yes*, we decide *No*; otherwise, we decide *Yes*. *(2 marks)*

The resulting decision tree is shown below.



(b) There are 200 states, 100 states for the positions in the maze when the robot has not yet picked up the treasure, and 100 states for the positions in the maze where the robot has. If we do not have separate states for being in the same position in the maze but with or without the treasure then the environment is no longer Markov. *(2 marks)* Actions are moving left, right, up and down and picking up the treasure. *(2 marks)* There are 5 actions, so there are $200 \times 5 = 1000$ state-action pairs. *(1 mark)*

(c) Augmenting the reward function would speed up learning, since the reward function would be more informative and help guide the agent's behaviour earlier. *(2 marks)* However, we take the risk of modifying the solutions found by the agent, if what we change the reward to does not exactly reflect what we want the agent to do. *(3 marks)*

6. Philosophy

(a) The *Weak AI claim* is the claim that a computer system can be built that acts intelligently. *(1 mark)* The *Strong AI claim* is the claim that such a machine would have a mind and in fact be thinking. *(1 mark)* The first claim is just about the *behaviour (1 mark)* of a system, whereas the second is about whether a computer can actually think, or whether it would simply be a *simulation* of thinking. *(2 marks)*.

Comment form

We welcome any comments you may have on the materials which are sent to you as part of your study pack. Such feedback from students helps us in our effort to improve the materials produced for the University of London.
If you have any comments about this guide, either general or specific (including corrections, non-availability of Essential readings, etc.), please take the time to complete and return this form.

Title of this subject guide: .....

.....

Name .....

Address .....

.....

Email .....

Student number .....

For which qualification are you studying? .....

Comments

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Please continue on additional sheets if necessary.

Date: .....

Please send your completed form (or a photocopy of it) to:
Publishing Manager, Publications Office, University of London, Stewart House, 32 Russell Square, London WC1B 5DN, UK.