Künstliche Intelligenz

Ein moderner Ansatz

4., aktualisierte Auflage

Stuart Russell
Peter Norvig

Pearson

EXTRAS ONLINE

Künstliche Intelligenz

Künstliche Intelligenz

Ein moderner Ansatz

4., aktualisierte Auflage

Stuart Russell
Peter Norvig

Pearson

Bibliografische Information der Deutschen Nationalbibliothek

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.dnb.de> abrufbar.

Bildnachweise Cover:
Alan Turing – Science History Images/Alamy Stock Photo
Ada Lovelace – Pictorial Press Ltd/Alamy Stock Photo
Autonomes Auto – Andrey Suslov/Shutterstock
Roboter Atlas – Boston Dynamics, Inc.
Berkeley-Campus und Golden Gate Bridge – Ben Chu/Shutterstock
Schachbrett mit Schachfiguren – Titania/Shutterstock
G. Kasparow – KATHY WILLENS/AP Images

10 9 8 7 6 5 4 3 2 1

27 26 25 24 23

ISBN 978-3-86894-430-3 (Buch)
ISBN 978-3-86326-326-3 (E-Book)

St.-Martin-Straße 82, D-81541 München

www.pearson.de
A part of Pearson plc worldwide

Programmleitung: Birger Peil, bpeil@pearson.de
Fachlektorat: Prof. Dr. Gabriele Kern-Isberner, TU Dortmund
Übersetzung: Petra Alm; Katharina Pieper
Sprachkorrektorat: Katharina Pieper, Berlin
Bildbearbeitung: Elisabeth Prümm, epruemm@pearson.de
Satz: le-tex publishing services GmbH, Leipzig
Druck und Verarbeitung: Neografia a.s., Martin-Priekopa

Printed in Slovakia

Für Loy, Gordon, Lucy, George und Isaac – S.J.R.

Für Kris, Isabella und Juliet – P.N.

Inhaltsverzeichnis

IV Unsicheres Wissen und Schlussfolgern

Vorwort

Künstliche Intelligenz (KI) ist ein weites Feld und dieses Buch entsprechend umfangreich. Wir haben versucht, der ganzen Bandbreite des Themas gerecht zu werden, die von Logik, Wahrscheinlichkeit und stetiger Mathematik über Wahrnehmung, Schlussfolgern, Lernen und Handeln bis hin zu Fairness, Vertrauen, Gemeinwohl und Sicherheit reicht. Begleitet werden unsere Ausführungen von anschaulichen Anwendungsbeispielen, sei es zu mikroelektronischen Geräten, Robotermobilen zur Erkundung ferner Planeten bis hin zu Onlinediensten mit Milliarden von Benutzern.

Der Untertitel dieses Buchs lautet „Ein moderner Ansatz“, was ausdrücken soll, dass wir die Geschichte aus aktueller Perspektive erzählen wollen. Wir fassen alle bisherigen Erkenntnisse in einem Rahmenwerk zusammen, indem wir frühere Arbeiten auf der Basis der heutigen Vorstellungen und Terminologie neu formulieren. Wir entschuldigen uns bei allen, deren Teilgebiete dadurch weniger eindeutig zu erkennen sind.

Neu in dieser Auflage

Diese Auflage spiegelt die Änderungen in der KI seit der letzten englischen Ausgabe im Jahr 2010 wider:

- Aufgrund der zunehmenden Verfügbarkeit von Daten, Rechenressourcen und neuen Algorithmen konzentrieren wir uns mehr auf maschinelles Lernen als auf die händische Wissensmodellierung.
- Deep Learning, probabilistische Programmierung und Multiagentensysteme werden in jeweils einem eigenen Kapitel ausführlich behandelt.
- Die Themen natürliche Sprachverarbeitung, Robotik und Computer Vision wurde überarbeitet, um dem Einfluss von Deep Learning Rechnung zu tragen.
- Das Robotik-Kapitel umfasst nun auch Roboter, die mit Menschen interagieren, sowie den Einsatz von Reinforcement Learning in der Robotik.
- Früher bestand das Ziel der KI darin, Systeme zu entwickeln, mit denen sich der erwartete Nutzen maximieren lässt, wobei die spezifischen Nutzeninformationen – das Ziel – von den menschlichen Entwicklern des Systems vorgegeben werden. Inzwischen gehen wir nicht mehr davon aus, dass das Ziel feststeht und dem KI-System bekannt ist, sondern dass das System durchaus unsicher hinsichtlich der wahren Ziele der Menschen sein kann, in deren Auftrag es arbeitet. Es muss lernen, was zu maximieren ist, und muss auch dann angemessen funktionieren, wenn es sich über das Ziel im Unklaren ist.
- Wir gehen stärker auf die Auswirkungen von KI auf die Gesellschaft ein, einschließlich so wichtiger Fragen wie Ethik, Fairness, Vertrauen und Sicherheit.
- Wir haben die Übungen vom Ende jedes Kapitels auf eine Website verschoben. Auf diese Weise können wir die Übungen kontinuierlich ergänzen, aktualisieren und verbessern, um den Anforderungen der Dozentinnen und Dozenten gerecht zu werden und die Fortschritte auf dem Gebiet und bei den KI-bezogenen Software-Tools zu berücksichtigen.
- Insgesamt sind etwa 25 % des Materials in diesem Buch komplett neu. Die restlichen 75 % wurden weitgehend neu geschrieben, um das Thema möglichst einheitlich zu präsentieren. 22 % der Literaturangaben in dieser Ausgabe beziehen sich auf Arbeiten, die nach 2010 veröffentlicht wurden.

Überblick über das Buch

Das wichtigste verbindende Element ist das Konzept eines **intelligenten Agenten**. Wir definieren KI als das Studium von Agenten, die Perzepte aus der Umgebung empfangen und Aktionen ausführen. Jeder dieser Agenten implementiert eine Funktion, die Perzeptfolgen auf Aktionen abbildet, und wir beschreiben verschiedene Möglichkeiten, diese Funktionen zu repräsentieren, z. B. reaktive Agenten, Echtzeitplaner, entscheidungstheoretische Systeme und Deep-Learning-Systeme. Wir legen den Schwerpunkt auf das Lernen sowohl als Konstruktionsmethode für kompetente Systeme als auch als Möglichkeit, die Reichweite des Entwicklers in unbekannte Umgebungen zu erweitern. Wir behandeln Robotik und Vision nicht als unabhängig definierte Probleme, sondern als Probleme, die beim Erreichen von Zielen auftreten. Wir legen das Augenmerk auf die Bedeutung der Aufgabenumgebung bei der Festlegung des geeigneten Agentenentwurfs.

Unser primäres Ziel ist es, die *Konzepte und Ideen* zu vermitteln, die sich in den letzten siebzig Jahren der KI-Forschung und den letzten zwei Jahrtausenden verwandter Arbeiten herausgebildet haben. Wir haben versucht, diese Ideen nicht zu formal zu präsentieren, ohne dass dies jedoch zu Lasten der Genauigkeit geht. Wir haben mathematische Formeln und Pseudocode-Algorithmen verwendet, um die wichtigsten Ideen zu konkretisieren. Eine Beschreibung der mathematischen Begriffe und Notation finden Sie in Anhang A und unseren Pseudocode beschreiben wir in Anhang B.

Dieses Buch ist in erster Linie für den Einsatz in einem Grundkurs gedacht. Das Buch besteht aus 28 Kapiteln, von denen jedes etwa eine Woche Vorlesungszeit erfordert, sodass das Durcharbeiten des gesamten Buchs sich über einen zweisemestrigen Grundkurs erstreckt. In einem einsemestrigen Kurs können, orientiert an den Interessen der Dozentinnen und Dozenten sowie der Studentinnen und Studenten, ausgewählte Kapitel durchgearbeitet werden. Das Buch kann auch im Masterstudium eingesetzt werden (vielleicht indem einige der Primärquellen hinzugezogen werden, die in den bibliografischen Anmerkungen vorgeschlagen wurden). Ebenso eignet es sich zum Selbststudium oder als Nachschlagewerk.

Im gesamten Buch sind *wichtige Punkte* mit einem Dreieckssymbol am Rand gekennzeichnet. Wichtige Begriffe werden **fett** gedruckt, einige sind zusätzlich noch **farbig** – hier wird der Begriff in der Regel eingeführt bzw. definiert. Das Buch beinhaltet einen umfassenden Index, ein Namensregister und ein umfangreiches Literaturverzeichnis.

Die einzige Voraussetzung ist eine gewisse Kenntnis der grundlegenden Konzepte der Informatik (Algorithmen, Datenstrukturen, Komplexität) auf Anfängerniveau. Für einige der Themen sind grundlegende Kenntnisse der Analysis und der linearen Algebra von Vorteil.

Online-Materialien

Online-Materialien sind über `pearsonhighered.com/cs-resources` oder auf der Website der Autoren zum Buch, `aima.cs.berkeley.edu`, verfügbar. Dort finden Sie:

- Übungen, Programmierprojekte und Forschungsprojekte: Diese befinden sich nicht mehr am Ende eines jeden Kapitels, sondern sind nur noch online verfügbar. Im Buch selber verweisen wir auf eine Online-Übung über ihre Nummerierung wie „Übung 6.6“. Mit den Anweisungen auf der Website können Sie die Übungen nach Thema oder Nummer suchen.
- Implementierungen der Algorithmen im Buch sind in Python, Java und anderen Programmiersprachen im Online-Code-Repository erhältlich (derzeit gehostet auf `github.com/aimacode`).

- eine Liste von über 1.500 Lehrinstituten, die das Buch verwendet haben, viele mit Links zu online verfügbaren Kursmaterialien und Lehrplänen;
- ergänzendes Material in englisch und Links für Studentinnen und Studenten sowie Lehrkräfte.

Cover

Das Cover zeigt einen Teil der Endstellung aus der entscheidenden sechsten Partie des Schachspiels von 1997, in der das Programm Deep Blue Garry Kasparow (mit Schwarz) besiegte, wo zum ersten Mal ein Weltmeister von einem Computer in einem Schachspiel geschlagen wurde. Kasparow ist oben abgebildet. Rechts von ihm ist eine Schlüsselstellung aus der zweiten Partie des historischen Go-Spiels zwischen dem ehemaligen Weltmeister Lee Sedol und dem Programm ALPHAGO von DeepMind zu sehen. Zug 37 von ALPHAGO verstieß gegen die jahrhundertealte Go-Tradition und wurde von menschlichen Experten sofort als peinlicher Fehler angesehen, der sich jedoch als Gewinnzug herausstellte. Oben links ist der von Boston Dynamics entwickelte humanoide Roboter Atlas zu sehen. Zwischen Ada Lovelace, der ersten Computerprogrammiererin der Welt, und Alan Turing, der mit seinen grundlegenden Arbeiten die künstliche Intelligenz definierte, ist die Darstellung eines selbstfahrenden Autos zu sehen, das seine Umgebung wahrnimmt. Hinter dem Schachbrett befindet sich ein probabilistisches Programmiermodell, das von der UN-Organisation zum Vertrag über das „umfassende Verbot von Nuklearversuchen zur Erkennung von Nuklearexplosionen aus seismischen Signalen“ verwendet wird. (In der englischen Originalausgabe gibt es noch weitere Bilder, die wir hier leider nicht aufnehmen konnten: Am unteren Rand des Schachbretts ist ein Roboter der Mars-Exploration-Rover-Mission abgebildet sowie eine Statue von Aristoteles, dem Pionier der Logik; seinen Planungsalgorithmus aus *De Motu Animalium* kann man hinter den Namen der Autoren sehen.)

Danksagungen

Es braucht ein globales Dorf, um ein Buch zu machen. Über 600 Personen haben Teile des Buchs gelesen und Verbesserungsvorschläge gemacht. Die vollständige Liste finden Sie auf `aima.cs.berkeley.edu/ack.html` – ihnen allen gilt unser Dank. Wir haben hier nur Platz, um einige besonders wichtige Mitwirkende zu erwähnen. Zunächst seien die beitragenden Verfasser genannt:

- Judea Pearl (Abschnitt 13.5, Kausale Netze);
- Vikash Mansinghka (Abschnitt 15.4, Programme als Wahrscheinlichkeitsmodelle);
- Michael Wooldridge (Kapitel 18, Entscheidungen in Multiagentenumgebungen);
- Ian Goodfellow (Kapitel 21, Deep Learning);
- Jacob Devlin und Mei-Wing Chang (Kapitel 24, Deep Learning für die Verarbeitung natürlicher Sprache);
- Jitendra Malik und David Forsyth (Kapitel 25, Computer Vision);
- Anca Dragan (Kapitel 26, Robotik).

Dann einige Schlüsselrollen:

- Cynthia Yeung und Malika Cantor (Projektleitung);
- Julie Sussman und Tom Galloway (Korrektorat und redaktionelle Bearbeitung);

- Omari Stephens (Illustrationen);
- Tracy Johnson (Lektorat);
- Erin Ault und Rose Kernan (Cover und Farbumsetzung);
- Nalin Chhibber, Sam Goto, Raymond de Lacaze, Ravi Mohan, Ciaran O'Reilly, Amit Patel, Dragomir Radiv und Samagra Sharma (Entwicklung des Onlinecodes und Mentoring);
- Studentinnen und Studenten des Google Summer of Code (Entwicklung des Onlinecodes).

Stuarts Dank gilt seiner Frau Loy Sheflott für ihre endlose Geduld und grenzenlose Weisheit. Er hofft, dass Gordon, Lucy, George und Isaac dieses Buch bald lesen werden, nachdem sie ihm verziehen haben, dass er so lange daran gearbeitet hat. RUGS (Russell's Unusual Group of Students) waren wie immer außergewöhnlich hilfreich.

Peters Dank gilt seinen Eltern (Torsten und Gerda), die ihn zu diesem Projekt ermutigt haben, sowie seiner Frau (Kris), seinen Kindern (Bella und Juliet), seinen Kollegen, seinem Chef und seinen Freunden, die ihn während der langen Stunden des Schreibens und Neuschreibens ermutigt und erduldet haben.

Vorwort zur deutschen Auflage

Die Künstliche Intelligenz (KI) hat seit ihrem Geburtsjahr 1956 schon viele Höhen und Tiefen erlebt. Einer enthusiastischen Begeisterungswelle über neue Techniken und Methoden folgte in der Regel eine herbe Ernüchterungsphase – zu groß waren die Erwartungen, zu tief die Enttäuschung über Fehlschläge. Aktuell befindet sich die KI wieder auf einem Höhenflug und vieles spricht dafür, dass sich dieser Trend dieses Mal zumindest auf einem hohen Plateau stabilisiert: Die Erfolge des Deep Learning bei der Verarbeitung von Bildern und natürlicher Sprache sind mehr als überzeugend, ChatGPT begeistert sogar Experten und smarte intelligente Geräte prägen längst unseren Alltag und unser gesellschaftliches Zusammenleben.

Es scheint also gar nicht mehr ohne KI zu gehen, aber was genau bedeutet das für den einzelnen und die Gesellschaft? Wie lassen sich Fehlschläge, die auch dieses Mal unweigerlich bei der Anwendung von neuen KI-Techniken auftreten, perspektivisch einschätzen? Wie können wir die Vorteile und „Segnungen" der KI so einsetzen und nutzen, dass sie wirklich die Lebensqualität der Menschen verbessern? Es wird viel von „explainable AI" (erklärbare KI) und „human-centred AI" (auf den Menschen zentrierte KI) gesprochen, um den Menschen explizit in KI-Vorgänge einzubinden. Das sind ganz zentrale Aspekte, damit KI-Innovationen auch als Fortschritt von allen Beteiligten wahrgenommen werden.

Auf der Seite der menschlichen Akteure gilt es aber auch, ein besseres Verständnis für grundlegende KI-Techniken aufzubauen, um deren Nutzen und Grenzen besser beurteilen zu können. Eine solche Beurteilungskompetenz ist unverzichtbar, wenn KI tatsächlich ein wertgeschätzter Teil unseres beruflichen und privaten Lebens werden soll. Für viele Nicht-Experten ist die KI immer noch so etwas wie eine magische Komponente, die auf wunderbare Weise erstaunliche Dinge leistet. „Das macht die KI" hört man dann oft, wobei das sowohl bewundernd wie auch vorwurfsvoll klingen kann. Es soll hier gar nicht bezweifelt werden, dass die Wissenschaften allgemein schon viele wunderbare Erkenntnisse zutage gefördert haben und dass die Ergebnisse des Deep Learning auch eine Herausforderung an das Verständnisvermögen der Experten darstellen. Aber vieles lässt sich doch in seinen Grundzügen recht anschaulich auf einem Niveau erklären, das KI auch für Nicht-Experten zugänglich macht. Zum Beispiel ließe sich das Grundprinzip des Lernens von Entscheidungsbäumen aus Beispielen (hier in Kapitel 19 behandelt) mit einem einfachen Zählalgorithmus schon Schulkindern nahe bringen.

Und genau hier setzt dieses Buch an. Im Vorwort findet man den Hinweis, dass lediglich „gewisse Kenntnisse der grundlegenden Konzepte der Informatik auf Anfängerniveau“ erforderlich sind, um dieses Buch lesen zu können. Und das darf man getrost wörtlich nehmen. Das Buch führt auch in schwierige Themen überaus lesbar und anschaulich ein, vertieft zentrale Details bis auf die mathematische Ebene in verständlicher Weise und gibt einen hervorragenden Einblick in und Überblick über den Stand der KI-Forschung. So findet auch die KI-Expertin oder der KI-Experte hier ergiebiges Material für den Einstieg in Gebiete, die ihr oder ihm weniger vertraut sind.

Es war mir ein besonderes Vergnügen, dieses Buch im Rahmen des Fachlektorats zu lesen und mit dem hervorragenden Übersetzerinnen-Team Katharina Pieper und Petra Alm sowie dem Senior Publisher von Pearson Deutschland, Birger Peil, zusammenzuarbeiten. Nun bleibt mir noch, auch dem Leser viel Freude und vertiefte Einsichten in die KI bei der Lektüre dieses Buches zu wünschen.

Dortmund Gabriele Kern-Isberner

Über die Autoren

Stuart Russell wurde 1962 in Portsmouth, England, geboren. Er erhielt seinen B.A. mit Auszeichnung in Physik von der Universität Oxford im Jahr 1982 und seinen Ph.D. in Informatik von Stanford im Jahr 1986. Danach wechselte er an die Universität von Kalifornien in Berkeley, wo er als Professor und ehemaliger Lehrstuhlinhaber für Informatik, Direktor des Center for Human-Compatible AI und Inhaber des Smith-Zadeh-Lehrstuhls für Ingenieurwissenschaften arbeitet. Im Jahr 1990 erhielt er den Presidential Young Investigator Award der National Science Foundation und wurde 1995 mit dem Computers and Thought Award ausgezeichnet. Er ist Mitglied der American Association for Artificial Intelligence, der Association for Computing Machinery und der American Association for the Advancement of Science, Foschungsstipendiat des Wadham College, Oxford, und Andrew Carnegie Fellow. Von 2012 bis 2014 war er Preisträger des Chaire Blaise Pascal in Paris. Er hat über 300 Publikationen zu einem breiten Spektrum von Themen der künstlichen Intelligenz veröffentlicht. Zu seinen Büchern gehören unter anderem *The Use of Knowledge in Analogy and Induction*, *Do the Right Thing: Studies in Limited Rationality* (mit Eric Wefald) und *Human Compatible: Artificial Intelligence and the Problem of Control*.

Peter Norvig ist derzeit Forschungsleiter bei Google, Inc. und war zuvor als Leiter für die wichtigsten Algorithmen der Websuche verantwortlich. Er war Co-Dozent eines KI-Onlinekurses, für den sich 160.000 Studenten anmeldeten, und gab damit den Startschuss für die aktuelle Runde der vielzähligen offen zugänglichen Onlinekurse. Er war Leiter der Computational Sciences Division am NASA Ames Research Center, wo er für die Forschung und Entwicklung in den Bereichen künstliche Intelligenz und Robotik verantwortlich war. Er erhielt einen B.S. in angewandter Mathematik von der Brown-Universität und einen Ph.D. in Informatik von Berkeley. Er war Professor an der Universität von Südkalifornien und Fakultätsmitglied in Berkeley und Stanford. Er ist ein Mitglied der American Association for Artificial Intelligence, der Association for Computing Machinery, der American Academy of Arts and Sciences und der California Academy of Science. Seine weiteren Bücher lauten *Paradigms of AI Programming: Case Studies in Common Lisp*, *Verbmobil: A Translation System for Face-to-Face Dialog* und *Intelligent Help Systems for UNIX*.

Beide Autoren teilten sich im Jahr 2016 den ersten AAAI/EAAI Outstanding Educator Award.

1 Einleitung

In diesem Kapitel wollen wir erklären, warum wir die Künstliche Intelligenz für ein absolut lohnenswertes Forschungsgebiet halten. Wir wollen versuchen zu definieren, worum es sich dabei genau handelt – eine sinnvolle Maßnahme, bevor wir wirklich anfangen.

Wir bezeichnen uns selbst als *Homo sapiens* – den weisen Menschen –, weil unsere **Intelligenz** so wichtig ist für uns. Seit Tausenden von Jahren versuchen wir zu verstehen, *wie wir denken und handeln* – das heißt, wie unser Gehirn, eine Handvoll Materie, eine Welt wahrnehmen, verstehen, vorhersagen und manipulieren kann, die viel größer und komplizierter ist als es selbst. Das Gebiet der **Künstlichen Intelligenz** beschäftigt sich nicht nur mit dem Verstehen, sondern auch mit dem *Erstellen* intelligenter Einheiten – Maschinen, die berechnen können, wie sie in einer Vielzahl neuartiger Situationen effektiv und zuverlässig handeln können.

In Umfragen wird die KI regelmäßig als eines der interessantesten und am schnellsten wachsenden Felder eingestuft, und sie generiert bereits über eine Billion US-Dollar Umsatz pro Jahr. Der KI-Experte Kai-Fu Lee prognostiziert, dass die Auswirkungen der KI „größer als alles andere in der Geschichte der Menschheit" sein werden. Außerdem sind die intellektuellen Grenzen der KI weit offen. Während ein Student einer älteren Wissenschaft wie der Physik das Gefühl haben könnte, dass die besten Ideen bereits von Galileo, Newton, Curie, Einstein und anderen großen Denkern entdeckt wurden, bietet die KI einem Vollzeit-Superhirn noch viele Möglichkeiten.

Die KI umfasst momentan eine Vielzahl von Teilgebieten, die von allgemeinen Bereichen (Lernen, Schlussfolgerungen, Wahrnehmung usw.) bis zu spezifischen Bereichen wie Schach spielen, mathematische Theoreme beweisen, Poesie verfassen, Auto fahren oder Krankheiten diagnostizieren reichen. Die KI ist für jede intellektuelle Aufgabe relevant – sie ist wirklich ein universelles Gebiet.

1.1 Was ist KI?

Wir haben behauptet, die KI sei interessant, aber wir haben nicht gesagt, was KI *ist*. In der Vergangenheit haben Forscher mehrere verschiedene Ansätze der KI verfolgt. Einige haben Intelligenz in Bezug auf die Wiedergabetreue *menschlicher* Leistung definiert, während andere eine abstrakte, formale Definition von Intelligenz in der Form von **Rationalität** bevorzugen – grob gesagt, das „Richtige" zu tun. Auch der Gegenstand selbst variiert: Einige betrachten Intelligenz als eine Eigenschaft interner *Denkprozesse* und *Schlussfolgerungen*, während andere sich auf intelligentes *Verhalten* konzentrieren, eine externe Charakterisierung.[1]

Aus diesen beiden Dimensionen – intuitiv versus rational und Denken versus Verhalten – ergeben sich vier mögliche Kombinationen und für alle vier gab es Anhänger und Forschungsprogramme. Die verwendeten Methoden sind notwendigerweise unterschiedlich: Das Streben nach menschenähnlicher Intelligenz muss teilweise eine empirische Wissenschaft sein, die mit der Psychologie verwandt ist und Beobachtungen und Hypothesen über das tatsächliche menschliche Verhalten und die Denkprozesse beinhaltet; ein rationalistischer Ansatz hingegen beinhaltet eine Kombination aus Mathematik und Ingenieurwesen und ist mit Statistik, Kontrolltheorie und Wirtschaft verbunden. Die verschiedenen Gruppen haben sich gegenseitig sowohl diffamiert als auch unterstützt. Schauen wir uns die vier Ansätze genauer an.

1 In der Öffentlichkeit werden manchmal die Begriffe „künstliche Intelligenz" und „maschinelles Lernen" verwechselt. Maschinelles Lernen ist ein Teilgebiet der KI, das sich mit der Fähigkeit beschäftigt, die Leistung auf Basis von Erfahrungen zu verbessern. Einige KI-Systeme nutzen Methoden des maschinellen Lernens, um Kompetenz zu erlangen, manche aber auch nicht.

1.1.1 Menschliches Handeln: der Ansatz mit dem Turing-Test

Der von Alan Turing (1950) vorgeschlagene **Turing-Test** wurde als Gedankenexperiment konzipiert, das die philosophische Unbestimmtheit der Frage „Kann eine Maschine denken?“ umgehen sollte. Ein Computer besteht den Test, wenn ein menschlicher Fragesteller, der einige schriftlichen Fragen stellt, nicht erkennen kann, ob die schriftlichen Antworten von einem Menschen oder von einem Computer stammen. In Kapitel 27 werden die Details des Tests besprochen und wir gehen der Frage nach, ob ein Computer wirklich intelligent wäre, wenn er den Test bestünde. Für den Moment halten wir fest, dass die Programmierung eines Computers, der den Turing-Test im strengen Sinne bestehen soll, eine Menge Arbeit mit sich bringt. Der Computer müsste die folgenden Fähigkeiten besitzen:

- **Verarbeitung natürlicher Sprache** (NLP, *Natural Language Processing*), um erfolgreich in einer menschlichen Sprache zu kommunizieren;
- **Wissensrepräsentation**, um zu speichern, was er weiß oder hört;
- **automatisches Schlussfolgern**, um Fragen anhand gespeicherter Informationen zu beantworten und neue Schlussfolgerungen zu ziehen;
- **maschinelles Lernen**, um sich an neue Gegebenheiten anzupassen sowie Muster zu erkennen und zu extrapolieren.

Turing hielt die *physische* Simulation einer Person für unnötig, um Intelligenz zu demonstrieren. Andere Forscher haben jedoch einen **totalen Turing-Test** vorgeschlagen, der die Interaktion mit Objekten und Menschen in der realen Welt erfordert. Um den totalen Turing-Test zu bestehen, braucht ein Rechner außerdem

- **Computer Vision und Spracherkennung**, um die Welt wahrzunehmen;
- **Robotik**, um Objekte zu manipulieren und sich zu bewegen.

Diese sechs Disziplinen bilden den größten Teil der KI. Dennoch haben sich KI-Forscher nur selten am Turing-Test orientiert, weil sie glauben, es sei wichtiger, die zugrunde liegenden Prinzipien der Intelligenz zu untersuchen. Schließlich war auch die Suche nach dem „künstlichen Fliegen“ erst erfolgreich, als Ingenieure und Erfinder aufhörten, Vögel zu imitieren, und stattdessen anfingen, Windkanäle einzusetzen und sich mit Aerodynamik zu beschäftigen. In Lehrbüchern der Luftfahrttechnik wird das Ziel ihres Fachgebiets eben nicht definiert als „Bau von Maschinen, die genau wie Tauben fliegen, sodass sie sogar andere Tauben täuschen können“.

1.1.2 Menschliches Denken: der Ansatz der kognitiven Modellierung

Wenn wir sagen, ein Programm denkt wie ein Mensch, so müssen wir wissen, wie Menschen denken. Wir haben drei Möglichkeiten, etwas über das menschliche Denken zu lernen:

- **Introspektion** – der Versuch, unsere eigenen Gedanken zu erfassen, während sie vorbeiziehen;
- **psychologische Experimente** – das Beobachten einer handelnden Person;
- **Neuroimaging** – das Beobachten des aktiven Gehirns.

Sobald wir eine hinreichend präzise Theorie des Verstands haben, wird es möglich, diese Theorie als Computerprogramm auszudrücken. Stimmen die Eingaben und Ausgaben des Programms mit dem entsprechenden menschlichen Verhalten überein, dann ist dies ein Beleg dafür, dass einige der Mechanismen des Programms auch beim Menschen funktionieren könnten.

Allen Newell und Herbert Simon, die den GPS – den „General Problem Solver“ (Newell und Simon, 1961) entwickelten, begnügten sich zum Beispiel nicht damit, dass ihr Programm Probleme korrekt löste. Es ging ihnen vielmehr darum, die Abfolge und das Timing der Schlussfolgerungsschritte mit denen menschlicher Probanden zu vergleichen, die dieselben Probleme lösten. Das interdisziplinäre Feld der **Kognitionswissenschaft** bringt Computermodelle aus der KI und experimentelle Techniken aus der Psychologie zusammen, um exakte und überprüfbare Theorien des menschlichen Verstands zu konstruieren.

Die Kognitionswissenschaft ist ein an sich faszinierendes Gebiet, das mehrerer Lehrbücher und mindestens einer Enzyklopädie würdig ist (Wilson und Keil, 1999). Wir werden gelegentlich auf Ähnlichkeiten oder Unterschiede zwischen KI-Techniken und menschlicher Kognition hinweisen. Die eigentliche Kognitionswissenschaft basiert jedoch notwendigerweise auf experimentellen Untersuchungen an „echten“ Menschen oder Tieren. Dies überlassen wir aber anderen Büchern, da wir davon ausgehen, dass der Leser für seine Experimente nur über einen Rechner verfügt.

In den Anfängen der KI gab es häufig Verwechselungen zwischen KI- und kognitiven Ansätzen. Ein Autor konnte behaupten, dass ein Algorithmus für eine Aufgabe gut geeignet und *deshalb* ein gutes Modell für die menschliche Leistung sei, oder umgekehrt. Moderne Autoren unterscheiden zwischen den beiden Ansätzen – diese Unterscheidung hat sowohl der KI als auch der Kognitionswissenschaft eine schnellere Entwicklung ermöglicht. Die beiden Bereiche befruchten sich gegenseitig, vor allem im Gebiet der Computer Vision, das neurophysiologische Erkenntnisse in Berechnungsmodelle einbezieht. In jüngster Zeit hat die Kombination von Methoden des Neuroimagings mit Techniken des maschinellen Lernens zur Analyse dieser Daten zu den ersten Anfängen eines „Gedankenlesens“ geführt – das heißt, den semantischen Inhalt von Gehirntätigkeiten einer Person zu ergründen. Diese Fähigkeit könnte wiederum die Funktionsweise der menschlichen Kognition weiter erhellen.

1.1.3 Rationales Denken: der Ansatz der „Denkgesetze“

Der griechische Philosoph Aristoteles war einer der ersten, der versuchte, „richtiges Denken“ – also unwiderlegbare Prozesse für logische Schlussfolgerungen – zu formalisieren. Seine **Syllogismen** lieferten Muster für Argumentationsstrukturen, die bei korrekten Prämissen immer zu richtigen Schlussfolgerungen führten. Das kanonische Beispiel beginnt mit *Sokrates ist ein Mensch* und *Alle Menschen sind sterblich*, daraus folgert er: *Sokrates ist sterblich*. (Dieses Beispiel geht wahrscheinlich eher auf Sextus Empiricus als auf Aristoteles zurück.) Diese Denkgesetze sollten die Arbeitsweise des Verstands abbilden; ihr Studium begründete das Gebiet der **Logik**.

Die Logiker des 19. Jahrhunderts entwickelten eine präzise Notation für Aussagen über Objekte in der Welt sowie den Beziehungen zwischen ihnen (im Gegensatz zur gewöhnlichen arithmetischen Notation, die nur Aussagen über *Zahlen* zulässt). Bis 1965 konnten Programme im Prinzip *jedes* lösbare Problem lösen, das in logischer Notation beschrieben war. Die sogenannte **logizistische** Tradition innerhalb der Künstlichen Intelligenz hofft, auf solchen Programmen aufbauen zu können, um intelligente Systeme zu schaffen.

Logik, wie sie konventionell verstanden wird, setzt ein Wissen über die Welt voraus, das *sicher* ist – eine Bedingung, die in der Realität nur selten erreicht wird. Wir kennen die Regeln etwa der Politik oder der Kriegsführung einfach nicht in der gleichen Weise wie die Regeln des Schachs oder der Arithmetik. Die **Wahrscheinlichkeitstheorie** füllt diese Lücke, indem sie stringentes Schlussfolgern mit unsicheren Informationen ermöglicht. Im Prinzip erlaubt sie die Konstruktion eines umfassenden Modells des rationalen Denkens, das von bloßen Wahrnehmungsinformationen zu einem Verständnis der Funktionsweise der Welt und zu Vorhersagen über die Zukunft führt. Was es nicht kann, ist, intelligentes *Verhalten* zu erzeugen. Dafür brauchen wir eine Theorie des rationalen Handelns. Rationales Denken allein reicht nicht aus.

1.1.4 Rationales Handeln: der Ansatz der rationalen Agenten

Ein **Agent** ist einfach etwas, das handelt (*Agent* kommt vom lateinischen *agere* – agieren, tun, handeln). Natürlich tun alle Computerprogramme etwas, doch von Computeragenten erwartet man, dass sie mehr tun: autonom agieren, ihre Umgebung wahrnehmen, über einen längeren Zeitraum Bestand haben, sich an Veränderungen anpassen sowie Ziele erstellen und verfolgen. Ein **rationaler Agent** handelt so, dass er das beste Ergebnis bzw. bei Unsicherheit das beste erwartete Ergebnis erzielt.

Beim KI-Ansatz der „Denkgesetze" lag die Betonung auf korrekten Schlussfolgerungen. Korrekte Schlüsse zu ziehen, ist manchmal *Teil* des rationalen Agenten, denn eine Möglichkeit, rational zu handeln, besteht gerade darin zu folgern, dass eine bestimmte Handlung die beste ist, und dann danach zu handeln. Andererseits gibt es Wege, rational zu handeln, ohne Schlussfolgerungen zu ziehen. Zum Beispiel ist das Zurückweichen vor einer heißen Herdplatte eine Reflexhandlung, die normalerweise erfolgreicher ist als eine langsamere Handlung, die nach sorgfältiger Überlegung ausgeführt wird.

Alle Fertigkeiten, die für den Turing-Test benötigt werden, erlauben es einem Agenten auch, rational zu handeln. Mithilfe von Wissensrepräsentation und Schlussfolgerungen können Agenten gute Entscheidungen treffen. Wir müssen in der Lage sein, verständliche Sätze in natürlicher Sprache zu bilden, um in einer komplexen Gesellschaft zurechtzukommen. Wir müssen nicht lernen, um Weisheit zu erlangen, sondern weil es unsere Fähigkeit verbessert, effektives Verhalten zu generieren, besonders unter neuen Umständen.

Der KI-Ansatz des rationalen Agenten hat zwei Vorteile gegenüber den anderen Konzepten. Erstens ist er allgemeiner als der Ansatz der „Denkgesetze", weil korrektes Schlussfolgern nur einer von mehreren möglichen Mechanismen ist, um Rationalität zu erreichen. Zweitens ist er zugänglicher für die wissenschaftliche Entwicklung. Der Standard der Rationalität ist mathematisch wohldefiniert und vollständig allgemein. Wir können oft ausgehend von dieser Spezifikation Agentenentwürfe ableiten, die den Standard nachweislich erreichen – etwas, das weitgehend unmöglich ist, wenn das Ziel darin besteht, menschliches Verhalten oder Denkprozesse zu imitieren.

Aus diesen Gründen hat sich das Konzept der rationalen Agenten in der KI-Geschichte allgemein durchgesetzt. In den ersten Jahrzehnten wurden rationale Agenten auf logischen Grundlagen aufgebaut und besaßen eindeutige Pläne, um bestimmte Ziele zu erreichen. Später ermöglichten Methoden, die auf der Wahrscheinlichkeitstheorie und dem maschinellen Lernen basierten, die Entwicklung von Agenten, die unter Unsicherheit Entscheidungen treffen konnten, um das beste erwartete Ergebnis zu erzielen. Kurz gesagt, *hat sich die KI auf die Untersuchung und Konstruktion von Agenten fokussiert, die* ***das Richtige tun***. Was als „das Richtige" zählt, wird durch das Ziel definiert, das wir dem Agenten vorgeben. Dieses allgemeine Paradigma ist so allgegenwärtig, dass wir es als **Standardmodell** bezeichnen könnten. Es ist nicht nur in der KI vorherrschend, sondern auch in der Kontrolltheorie, wo eine Kontrollinstanz (Controller) eine Kostenfunktion minimiert; im Bereich Operations Research, wo eine Strategie eine Gesamtbelohnung maximiert; in der Statistik, wo eine Entscheidungsregel eine Verlustfunktion minimiert; und in den Wirtschaftswissenschaften, wo ein Entscheidungsträger den Nutzen oder ein Maß für das Allgemeinwohl maximiert.

Wir müssen eine wichtige Präzisierung des Standardmodells vornehmen, um der Tatsache Rechnung zu tragen, dass perfekte Rationalität – immer genau die optimale Handlung zu wählen – in komplexen Umgebungen nicht erreichbar ist. Die Anforderungen an die Computerleistung sind einfach zu hoch. Die Kapitel 5 und 17 befassen sich mit dem Problem der **begrenzten Rationalität** – das heißt, angemessenes Handeln, wenn nicht genug Zeit für alle Berechnungen zur Verfügung steht, die man gerne durchführen würde. Perfekte Rationalität bleibt jedoch oft ein guter Ausgangspunkt für die theoretische Analyse.

1.1.5 Nutzbringende Maschinen

Das Standardmodell ist seit seinen Anfängen ein nützlicher Leitfaden für die KI-Forschung, aber es ist auf lange Sicht wahrscheinlich nicht das richtige Modell. Der Grund dafür ist, dass das Standardmodell davon ausgeht, dass wir der Maschine ein vollständig spezifiziertes Ziel liefern.

Bei einer künstlich definierten Aufgabe, wie z. B. Schach oder Berechnung des kürzesten Wegs, ist das Ziel direkt mit der Aufgabe verbunden – das Standardmodell ist also anwendbar. Wenn wir uns jedoch in die reale Welt begeben, wird es immer schwieriger, das Ziel vollständig und korrekt zu spezifizieren. Zum Beispiel könnte man bei der Entwicklung eines selbstfahrenden Autos denken, dass das Ziel darin besteht, den Bestimmungsort sicher zu erreichen. Aber jegliches Fahren auf einer Straße birgt ein Verletzungsrisiko, sei es durch Fehler anderer Fahrer, durch Geräteversagen usw. – ein striktes Sicherheitsziel müsste also dazu führen, in der Garage zu bleiben. Es gibt jedoch eine Abwägung zwischen dem Vorankommen in Richtung Bestimmungsort und dem Risiko einer Verletzung. Wie sollte man diese Abwägung treffen? Und inwieweit kann man dem Auto Aktionen erlauben, die andere Fahrer beeinträchtigen würden? Wie stark sollte das Auto seine Beschleunigung, Lenkung und Bremsen drosseln, um die Insassen nicht durchzuschütteln? Diese Art von Fragen sind a priori nur schwer zu beantworten. Besonders problematisch sind sie im allgemeinen Bereich der Mensch-Roboter-Interaktion, wofür das selbstfahrende Auto ein Beispiel ist.

Das Problem, eine Übereinstimmung zwischen unseren wahren Präferenzen und dem Ziel zu erreichen, das wir in die Maschine eingeben, ist das **Problem der Werteangleichung** (Value-Alignment-Problem): die der Maschine übergebenen Werte oder Ziele müssen mit denen des Menschen übereinstimmen. Wenn wir ein KI-System im Labor oder in einem Simulator entwickeln – wie es im Laufe der KI-Geschichte meistens der Fall war –, gibt es eine einfache Lösung für ein falsch spezifiziertes Ziel: das System zurücksetzen, das Ziel korrigieren und einen neuen Versuch starten. Mit der Entwicklung hin zu immer leistungsfähigeren intelligenten Systemen, die in der realen Welt eingesetzt werden, ist dieser Ansatz nicht mehr praktikabel. Ein System, das mit einem falschen Ziel eingesetzt wird, wird negative Auswirkungen haben. Und je intelligenter das System ist, desto negativer sind die Folgen.

Kommen wir noch einmal auf das scheinbar unproblematische Beispiel des Schachspiels zurück: Überlegen Sie, was passiert, wenn die Maschine intelligent genug ist, über die Grenzen des Schachbretts hinaus zu denken und zu handeln. Sie könnte in diesem Fall versuchen, ihre Gewinnchancen durch Tricks wie Hypnose oder Erpressung des Gegners zu erhöhen oder das Publikum zu bestechen, damit es raschelnde Geräusche in der Zeit macht, in der der Gegner nachdenkt.[2] Sie könnte außerdem versuchen, zusätzliche Rechenleistung für sich selbst zu kapern. *Diese Verhaltensweisen sind nicht „unintelligent“ oder „verrückt“ – sie sind eine logische Konsequenz daraus, wenn man Gewinnen als* einziges *Ziel der Maschine definiert.*

Es ist unmöglich vorherzusehen, auf welche Weisen sich eine Maschine, die ein bestimmtes Ziel verfolgt, falsch verhalten könnte. Das heißt, wir haben allen Grund anzunehmen, dass das Standardmodell unzureichend ist. Wir wollen keine Maschinen, die in dem Sinne intelligent sind, dass sie *ihre* Ziele verfolgen – wir möchten, dass sie *unsere* Ziele verfolgen. Wenn wir unsere Ziele nicht absolut korrekt auf die Maschine übertragen können, dann brauchen wir eine neue Zielformulierung – die Maschine soll unsere Ziele verfolgen, ist aber zwangsläufig *unsicher*, wie diese genau lauten. Weiß eine Maschine, dass sie das Gesamtziel nicht kennt, dann hat sie einen Anreiz, vorsichtig zu handeln, um Erlaubnis zu fragen, durch Beobachtung mehr über unsere Präferenzen zu erfahren und sich der menschlichen Steuerung zu unterstellen. Letztendlich wollen wir Agenten, die **nachweislich nutzbringend** für den Menschen sind. Wir werden auf dieses Thema in Abschnitt 1.5 zurückkommen.

2 In einem der ersten Bücher über Schach schrieb Ruy Lopez (1561): „Lege das Brett immer so, dass die Sonne in den Augen deines Gegners steht.“

1.2 Die Grundlagen der Künstlichen Intelligenz

In diesem Abschnitt geben wir einen kurzen geschichtlichen Abriss der Disziplinen, die mit Ideen, Sichtweisen und Techniken zur KI beigetragen haben. Wie jede Chronik konzentriert sich auch diese auf eine kleine Anzahl von Personen, Ereignisse und Ideen und ignoriert andere, die ebenfalls wichtig waren. Wir arrangieren unseren historischen Abriss um eine Reihe von Fragen herum. Keinesfalls möchten wir den Eindruck erwecken, die Disziplinen würden sich ausschließlich mit diesen Fragen befassen oder hätten alle auf die KI als ihre ultimative Erfüllung hingearbeitet.

1.2.1 Philosophie

- Kann man formale Regeln einsetzen, um gültige Schlüsse zu ziehen?
- Wie entsteht der Verstand aus einem physischen Gehirn?
- Woher stammt Wissen?
- Wie führt Wissen zum Handeln?

Aristoteles (384–322 v. Chr.) formulierte als Erster eine präzise Menge von Gesetzen, die den rationalen Teil des Geistes lenken. Er entwickelte ein informelles System von Syllogismen für korrektes Schlussfolgern, das es im Prinzip erlaubte, ausgehend von Prämissen mechanisch Schlüsse zu ziehen.

Ramon Llull (ca. 1232–1315) entwickelte ein System des logischen Denkens, das als *Ars Magna* (zu deutsch *Große Kunst*; 1305) veröffentlicht wurde. Llull versuchte, sein System mithilfe eines tatsächlichen mechanischen Geräts umzusetzen: einer Reihe von Papierrädern, die in verschiedene Kombinationen gedreht werden konnten.

Um das Jahr 1500 entwarf Leonardo da Vinci (1452–1519) eine mechanische Rechenmaschine, baute sie aber nie; neuere Rekonstruktionen zeigen, dass der Entwurf funktionstüchtig war. Die erste bekannte Rechenmaschine wurde um 1623 von dem deutschen Gelehrten Wilhelm Schickard (1592–1635) konstruiert. Blaise Pascal (1623–1662) baute 1642 die Pascaline und schrieb, dass sie „Wirkungen zeigt, die dem Denken näher kommen als alles, was Tiere vollbringen“. Gottfried Wilhelm Leibniz (1646–1716) baute unter anderem ein mechanisches Gerät, das Operationen mit Begriffen anstelle von Zahlen durchführen sollte, aber dessen Anwendungsbereich war eher begrenzt. Thomas Hobbes (1588–1679) schlug in seinem 1651 erschienen Buch *Leviathan* das Konzept einer denkenden Maschine vor, eines „künstlichen Tiers“, wie er es nannte, und argumentierte: „Denn was ist das Herz, wenn nicht eine Feder, was sind die Nerven, wenn nicht die vielen Stränge, und was nicht die Gelenke, wenn nicht die vielen Räder.“ Er schlug auch vor, das Denken einer numerischen Berechnung gleichzusetzen: „Denn ‚Vernunft‘ … ist nichts anderes als ‚Rechnen‘, das heißt Addieren und Subtrahieren.“

Es ist eine Sache zu sagen, dass der Verstand – zumindest teilweise – nach logischen oder numerischen Regeln arbeitet, und reale Systeme zu bauen, die einige dieser Regeln nachbilden. Eine andere Sache ist es zu behaupten, dass der Verstand selbst ein solches physisches System *ist*. René Descartes (1596–1650) lieferte die erste klare Diskussion zur Unterscheidung zwischen Geist und Materie. Er stellte fest, dass eine rein physische Auffassung des Geistes wenig Raum für den freien Willen zu lassen schien. Wenn der Geist ausschließlich von physischen Gesetzen beherrscht wird, dann hat er genauso wenig einen freien Willen wie ein Stein, der „beschließt“, nach unten zu fallen. Descartes war ein Verfechter des **Dualismus**. Er vertrat die Ansicht, dass es einen Teil des menschlichen Verstands (oder der Seele oder des Geistes) gibt, der außerhalb der Natur steht und nicht den Gesetzen der Physik unterliegt. Tiere hingegen besäßen diese duale Qualität nicht; sie könnten wie Maschinen behandelt werden.

Eine Alternative zum Dualismus ist der **Materialismus**, der davon ausgeht, dass die Funktionsweise des Gehirns den Geist nach den Gesetzen der Physik *bildet*. Der freie Wille ist einfach die Art und Weise, wie die wählende Entität die verfügbaren Alternativen wahrnimmt. Die Begriffe **Physikalismus** und **Naturalismus** werden ebenfalls verwendet, um diese Ansicht zu beschreiben, die im Gegensatz zum Übernatürlichen steht.

Angesichts eines physischen Geistes, der Wissen manipuliert, besteht das nächste Problem darin, die Quelle des Wissens zu bestimmen. Die Bewegung des **Empirismus**, beginnend mit Francis Bacons (1561–1626) *Novum Organum*[3], ist durch ein Diktum von John Locke (1632–1704) gekennzeichnet: „Nichts ist im Verstand, was nicht vorher in den Sinnen war."

David Hume (1711–1776) schlug in *Ein Traktat über die menschliche Natur* (Hume, 1739)[4] das vor, was heute als Prinzip der **Induktion** bekannt ist: Allgemeine Regeln werden erworben, indem man sich den Assoziationen zwischen ihren Elementen wiederholt aussetzt.

Aufbauend auf den Arbeiten von Ludwig Wittgenstein (1889–1951) und Bertrand Russell (1872–1970) entwickelte der berühmte Wiener Kreis (Sigmund, 2017), eine Gruppe von Philosophen und Mathematikern, die sich in den 1920er und 1930er Jahren in Wien trafen, die Doktrin des **logischen Positivismus**. Diese Lehre besagt, dass alles Wissen durch logische Theorien charakterisiert werden kann, die letztlich mit **Beobachtungssätzen** verbunden sind, die den Sinneseindrücken entsprechen – der logische Positivismus verbindet somit Rationalismus und Empirismus.

Die **Bestätigungstheorie** von Rudolf Carnap (1891–1970) und Carl Hempel (1905–1997) versuchte, den Erwerb von Wissen aus Erfahrung zu analysieren, indem der Grad der Überzeugung quantifiziert wurde, der logischen Sätzen zugewiesen werden sollte. Dieser Grad basiert auf der Verbindung der Sätze zu den Beobachtungen, die die Sätze bestätigen oder widerlegen. Carnaps Buch *Der logische Aufbau der Welt*[5] (1928) war vermutlich die erste Theorie, die den Verstand als Rechenprozess betrachtete.

Das letzte Element in der philosophischen Betrachtung des Verstands ist die Verbindung zwischen Wissen und Handeln. Diese Frage ist für die KI von entscheidender Bedeutung, weil Intelligenz sowohl Handeln als auch Denken erfordert. Außerdem können wir nur durch das Verständnis, wie Handlungen gerechtfertigt werden, verstehen, wie man einen Agenten baut, dessen Handlungen gerechtfertigt (oder rational) sind.

Aristoteles argumentierte in *De Motu Animalium* (*Über die Bewegung der Lebewesen*), dass Handlungen gerechtfertigt sind, wenn es eine logische Verbindung zwischen Zielen und dem Wissen über das Ergebnis der Handlung gibt:

> *Aber wie kann es sein, dass das Denken manchmal von Handeln begleitet wird und manchmal nicht, manchmal von Bewegung und manchmal nicht? Es scheint, als passiere fast dasselbe beim Denken und beim Schlussfolgern über sich unveränderliche Objekte. Doch in jenem Fall ist das Ziel eine spekulative Aussage ... während hier die Schlussfolgerung, die sich aus den beiden Prämissen ergibt, eine Handlung ist. ... Ich brauche Bekleidung; ein Mantel ist eine Bekleidung. Ich brauche einen Mantel. Was ich brauche, muss ich anfertigen; ich brauche einen Mantel. Ich muss einen Mantel anfertigen. Und die Schlussfolgerung „Ich muss einen Mantel anfertigen" ist eine Handlung.*

In der *Nikomachischen Ethik* (Buch III. 3, 1112b) geht Aristoteles weiter auf dieses Thema ein und schlägt einen Algorithmus vor:

> *Unsere Überlegung betrifft nicht das Ziel, sondern die Mittel, es zu erreichen. Der Arzt überlegt nicht, ob er heilen soll, der Redner nicht, ob er überzeugen soll, ... sondern nachdem man sich*

3 Das *Novum Organum* ist eine Aktualisierung von Aristoteles' *Organon*.
4 Originaltitel: *A Treatise of Human Nature*.
5 Originaltitel: *The Logical Structure of the World*.

> *ein Ziel gestellt hat, sieht man sich um, wie und durch welche Mittel es zu erreichen ist, wenn es durch verschiedene Mittel möglich scheint, sieht man zu, durch welches es am leichtesten und besten erreicht wird; und wenn es durch eines regelrecht verwirklicht wird, fragt man wieder,* wie *es durch dasselbe verwirklicht wird, und wodurch wiederum jenes, bis man zu der ersten Ursache gelangt, ... das, was bei der Analyse als Letztes herauskommt, ist bei der Verwirklichung durch die Handlung das Erste. Stößt man auf eine Unmöglichkeit, so steht man von der Sache ab, z. B. wenn sich Geldmittel erforderlich zeigen, die man nicht aufbringen kann. Erscheint die Sache aber möglich, so nimmt man sie in die Hand.*[6]

Der Algorithmus von Aristoteles wurde 2300 Jahre später von Newell und Simon in ihrem **General Problem Solver** implementiert. Wir würden es heute ein „gieriges Regressionsplanungssystem" nennen (siehe Kapitel 11). Methoden, die auf logischer Planung zum Erreichen bestimmter Ziele beruhen, dominierten die ersten Jahrzehnte der theoretischen Forschung in der KI.

Sich nur darauf zu konzentrieren, Ziele durch Handlungen zu erreichen, ist oft nützlich, aber manchmal ungeeignet. Gibt es z. B. verschiedene Wege, ein Ziel zu erreichen, so muss es eine Möglichkeit geben, zwischen diesen Wegen zu wählen. Noch entscheidender ist der Fall, wenn es vielleicht nicht möglich ist, ein Ziel mit Sicherheit zu erreichen, aber irgendeine Handlung trotzdem ausgeführt werden muss. Wie sollte man dann eine Entscheidung treffen? Antoine Arnauld (1662) analysierte das Konzept der rationalen Entscheidungen beim Glücksspiel und schlug eine quantitative Formel vor, mit deren Hilfe der erwartete Geldwert des Ergebnisses maximiert werden konnte. Später führte Daniel Bernoulli (1738) den allgemeineren Begriff des **Nutzens** ein, um den internen, subjektiven Wert eines Ergebnisses zu erfassen. Die moderne Vorstellung von rationaler Entscheidungsfindung unter Unsicherheit beinhaltet die Maximierung des erwarteten Nutzens, dieses Thema wird in Kapitel 16 behandelt.

In Fragen der Ethik und der öffentlichen Ordnung muss ein Entscheidungsträger die Interessen mehrerer Individuen berücksichtigen. Jeremy Bentham (1823) und John Stuart Mill (1863) unterstützten die Idee des **Utilitarismus**: das Konzept, dass die rationale Entscheidungsfindung, die auf der Maximierung des Nutzens basiert, für alle Bereiche menschlicher Aktivitäten gelten sollte, einschließlich der öffentlichen politischen Entscheidungen, die im Namen vieler Individuen getroffen werden. Der Utilitarismus ist eine spezifische Art des **Konsequenzialismus**: was richtig und falsch ist, wird durch die erwarteten Ergebnisse einer Handlung bestimmt.

Im Gegensatz dazu schlug Immanuel Kant im Jahr 1785 eine Theorie der regelbasierten oder **deontologischen Ethik** vor, in der „das Richtige tun" nicht von den Ergebnissen bestimmt wird, sondern von universellen sozialen Gesetzen, die erlaubte Handlungen regeln, wie z. B. „Du sollst nicht lügen" oder „Du sollst nicht töten". So könnte ein Utilitarist eine Notlüge benutzen, wenn das erwartete Gute das Schlechte überwiegt, aber ein Kantianer dürfte dies nicht tun, weil Lügen von Natur aus falsch sind. Mill erkannte den Wert von Regeln an, verstand sie aber als effiziente Entscheidungsprozeduren, die aus grundsätzlichen Überlegungen zu Konsequenzen zusammengestellt werden. Viele moderne KI-Systeme verfolgen genau diesen Ansatz.

1.2.2 Mathematik

- Was sind die formalen Regeln, um gültige Schlussfolgerungen zu ziehen?
- Was kann berechnet werden?
- Wie schlussfolgern wir mit unsicheren Informationen?

6 Übersetzung nach E. Rolfes (1910).

Die Philosophen umrissen einige grundlegenden Ideen der KI, doch der Sprung zu einer formalen Wissenschaft erforderte die mathematische Aufbereitung von Logik und Wahrscheinlichkeit sowie die Einführung eines neuen Zweigs der Mathematik: die Berechenbarkeit.

Das Konzept einer **formalen Logik** lässt sich bis zu den Philosophen des antiken Griechenlands, Indiens und Chinas zurückverfolgen, doch ihre mathematische Entwicklung begann eigentlich mit der Arbeit von George Boole (1815–1864), der die Details der Aussagenlogik (oder booleschen Logik) ausarbeitete (Boole, 1847). Im Jahr 1879 erweiterte Gottlob Frege (1848–1925) die boolesche Logik um Objekte und Relationen und schuf damit die heute verwendete Prädikatenlogik (oder Logik erster Ordnung).[7] Zusätzlich zu ihrer zentralen Rolle in der frühen Phase der KI-Forschung motivierte die Prädikatenlogik die Arbeiten von Gödel und Turing, die die Berechenbarkeit selbst zum Thema hatten, wie wir weiter unten noch ausführen werden.

Die **Wahrscheinlichkeitstheorie** kann als Verallgemeinerung der Logik auf Situationen mit unsicherer Informationslage gesehen werden – eine Überlegung, die für die KI von großer Bedeutung ist. Gerolamo Cardano (1501–1576) formulierte als Erster die Idee der Wahrscheinlichkeit und beschrieb sie in Bezug auf die möglichen Ergebnisse bei einem Glücksspiel. Im Jahr 1654 zeigte Blaise Pascal (1623–1662) in einem Brief an Pierre Fermat (1601–1665), wie man den Ausgang eines abgebrochenen Glücksspiels vorhersagen und den Spielern durchschnittliche Auszahlungen zuweisen kann. Die Wahrscheinlichkeitsrechnung wurde schnell zu einem unschätzbaren Teil der quantitativen Wissenschaften und half, mit unsicheren Messungen und unvollständigen Theorien umzugehen. Jakob Bernoulli (1654–1705, der Onkel von Daniel Bernoulli), Pierre Laplace (1749–1827) und andere entwickelten die Theorie weiter und führten neue statistische Methoden ein. Thomas Bayes (1702–1761) schlug eine Regel zum Aktualisieren von Wahrscheinlichkeiten angesichts neuer Informationen vor – diese Bayes'sche Regel ist ein wichtiges Werkzeug für KI-Systeme.

Die Formalisierung der Wahrscheinlichkeit, kombiniert mit der Verfügbarkeit von Daten, führte zur Entstehung der **Statistik** als neuem Fachgebiet. Eine der ersten Anwendungen war die Analyse von Daten der Londoner Sterbeverzeichnisse durch John Graunt im Jahr 1662. Ronald Fisher gilt als der erste moderne Statistiker (Fisher, 1922). Er brachte die Konzepte von Wahrscheinlichkeit, statistischer Versuchsplanung, Datenanalyse und Durchführen von Berechnungen zusammen – im Jahr 1919 bestand er darauf, dass er für seine Arbeit unbedingt eine mechanische Rechenmaschine namens MILLIONAIRE (die erste Rechenmaschine, die multiplizieren konnte) benötigte, obwohl die Anschaffungskosten für die Rechenmaschine sein Jahresgehalt überstiegen (Ross, 2012).

Die Geschichte des Rechnens ist so alt wie die Geschichte der Zahlen, aber als erster nicht trivialer **Algorithmus** gilt Euklids Algorithmus zur Berechnung des größten gemeinsamen Teilers zweier natürlicher Zahlen. Das Wort *Algorithmus* stammt von Muhammad ibn Musa al-Khwarizmi, einem Mathematiker des 9. Jahrhunderts, dessen Schriften auch die arabischen Ziffern und die Algebra nach Europa brachten. Boole und andere diskutierten Algorithmen für die logische Deduktion und im späten 19. Jahrhundert gab es Bestrebungen, allgemeine mathematische Beweise als logische Deduktion zu formalisieren.

Kurt Gödel (1906–1978) zeigte, dass es ein effektives Verfahren gibt, um jede wahre Aussage in der Prädikatenlogik von Frege und Russell zu beweisen, dass die Prädikatenlogik jedoch nicht das Prinzip der mathematischen Induktion erfassen kann, das zur Charakterisierung der natürlichen Zahlen notwendig ist. Im Jahr 1931 bewies Gödel, dass es Grenzen für die Deduktion gibt. Sein **Unvollständigkeitssatz** zeigte, dass es in jeder formalen Theorie, die so stark ist wie die Peano-Arithmetik (die elementare Theorie der natürlichen Zahlen), notwendigerweise wahre Aussagen gibt, die nicht innerhalb der Theorie bewiesen werden können.

7 Die von Frege vorgeschlagene Notation für die Prädikatenlogik – eine geheimnisvolle Kombination aus textuellen und geometrischen Merkmalen – hat sich nie durchgesetzt.

Dieses fundamentale Ergebnis kann auch dahingehend interpretiert werden, dass einige Funktionen auf den ganzen Zahlen nicht durch einen Algorithmus dargestellt werden können – das heißt, nicht berechnet werden können. Dies motivierte Alan Turing (1912–1954) zu dem Versuch, genau zu charakterisieren, welche Funktionen **berechenbar** sind – also durch ein effektives Verfahren berechnet werden können. Die Church-Turing-These schlägt vor, allgemein eine Funktion als berechenbar zu definieren, wenn sie von einer Turing-Maschine berechnet werden kann (Turing, 1936). Turing zeigte außerdem, dass es einige Funktionen gibt, die von keiner Turing-Maschine berechnet werden können. Zum Beispiel kann keine Maschine *im Allgemeinen* sagen, ob ein bestimmtes Programm bei einer gegebenen Eingabe eine Antwort zurückgibt oder unendlich läuft.

Obwohl die Berechenbarkeit wichtig für das Verständnis von Rechenoperationen ist, hatte der Begriff der **effizienten Machbarkeit** (*tractability*) einen noch größeren Einfluss auf die KI. Vereinfacht ausgedrückt wird ein Problem als nicht effizient machbar (*intractable*) bezeichnet, wenn die Zeit, die für die Lösung von Instanzen des Problems benötigt wird, exponentiell mit der Größe der Instanzen wächst. Die Unterscheidung zwischen polynomiellem und exponentiellem Komplexitätswachstum wurde erstmals Mitte der 1960er (Cobham, 1964; Edmonds, 1965) Jahre thematisiert. Sie ist wichtig, weil exponentielles Wachstum bedeutet, dass selbst mäßig große Instanzen nicht in einer angemessenen Zeit gelöst werden können.

Die Theorie der **NP-Vollständigkeit**, die auf Cook (1971) und Karp (1972) zurückgeht, bietet eine Grundlage für die Analyse der effizienten Machbarkeit von Problemen: Jede Problemklasse, auf die die Klasse der NP-vollständigen Probleme reduziert werden kann, ist wahrscheinlich nicht effizient machbar. (Obwohl es nicht bewiesen ist, dass NP-vollständige Probleme zwangsläufig nicht effizient machbar sind, gehen die meisten Theoretiker davon aus.) Diese Ergebnisse stehen im Gegensatz zu dem Optimismus, mit dem die Boulevardpresse die ersten Computer begrüßt hatte – „elektronische Superhirne“, die „schneller als Einstein“ wären! Trotz der steigenden Geschwindigkeit von Computern zeichnen sich intelligente Systeme durch sorgfältige Nutzung von Ressourcen und die notwendige Unvollkommenheit aus. Die Welt ist, salopp gesagt, eine *extrem* große Probleminstanz!

1.2.3 Wirtschaftswissenschaft

- Wie sollten wir Entscheidungen treffen, die mit unseren Präferenzen übereinstimmen?
- Wie sollten wir dies tun, wenn andere nicht mitmachen?
- Wie sollten wir dies tun, wenn der Gewinn weit in der Zukunft liegt?

Die Wirtschaftswissenschaft entstand 1776, als Adam Smith (1723–1790) sein Werk *Der Wohlstand der Nationen*[8] veröffentlichte. Smith schlug vor, Wirtschaftssysteme zu analysieren, die aus vielen einzelnen Akteuren bestehen, die jeweils ihre eigenen Interessen wahrnehmen. Smith war jedoch kein Verfechter der finanziellen Gier als moralische Position: Sein früheres Buch *Theorie der ethischen Gefühle*[9] (1759) beginnt mit dem Hinweis, dass die Sorge um das Wohlergehen anderer ein wesentlicher Bestandteil der Interessen eines jeden Individuums ist.

Die meisten Menschen denken, dass es in der Wirtschaftswissenschaft um Geld geht, und in der Tat befasste sich die erste mathematische Analyse von Entscheidungen unter Unsicherheit, die Formel für den maximalen Erwartungswert von Arnauld und Nicole (1662), mit dem Geldwert von Wetten. Daniel Bernoulli (1738) bemerkte, dass diese Formel für größere Geldbeträge, wie z. B. Investitionen in maritime Handelsexpeditionen, nicht gut zu funktionieren schien. Er schlug stattdessen ein Prinzip vor, das auf der Maximierung des erwarteten

8 Originaltitel: *An Inquiry into the Nature and Causes of the Wealth of Nations.*
9 Originaltitel: *The Theory of Moral Sentiments.*

Nutzens basiert, und erklärte die menschlichen Investitionsentscheidungen damit, dass der Grenznutzen einer zusätzlichen Geldmenge abnimmt, wenn man mehr Geld erwirbt.

Léon Walras (1834–1910) gab der Nutzentheorie eine allgemeinere Grundlage, indem er Präferenzen zwischen Spielen mit beliebigen Ergebnissen (nicht nur monetäre Ergebnisse) einführte. Die Theorie wurde von Ramsey (1931) und später von John von Neumann und Oskar Morgenstern in ihrem Buch *The Theory of Games and Economic Behavior* (1944) verbessert. In der Wirtschaftswissenschaft geht es nicht mehr nur um Geld, sondern um Wünsche und Präferenzen.

Die **Entscheidungstheorie**, die die Wahrscheinlichkeitstheorie mit der Nutzentheorie kombiniert, bietet einen einfachen und vollständigen Rahmen für individuelle (wirtschaftliche oder anderweitige) Entscheidungen, die unter Unsicherheit getroffen werden – das heißt, in Fällen, in denen probabilistische Beschreibungen die Umgebung des Entscheidungsträgers angemessen erfassen. Dies ist geeignet für „große“ Wirtschaftssysteme, in denen kein Agent die Handlungen anderer Agenten als Individuen berücksichtigen muss. Für „kleine“ Wirtschaftssysteme ähnelt die Situation eher einem **Spiel**: Die Handlungen eines Spielers können den Nutzen eines anderen signifikant beeinflussen (entweder positiv oder negativ). Die von Morgenstern und von Neumann entwickelte **Spieltheorie** (siehe auch Luce und Raiffa, 1957) beinhaltete das überraschende Ergebnis, dass ein rationaler Agent bei einigen Spielen eine Zufallsstrategie (oder die zumindest als zufällig erscheint) verfolgen sollte. Anders als die Entscheidungstheorie bietet die Spieltheorie keine eindeutige Vorschrift dafür, wie Aktionen auszuwählen sind. In der KI werden Entscheidungen, an denen mehrere Agenten beteiligt sind, unter dem Begriff **Multiagentensysteme** behandelt (Kapitel 18).

Wirtschaftswissenschaftler haben sich, von einigen Ausnahmen abgesehen, nicht mit der dritten der oben genannten Fragen befasst: wie man rationale Entscheidungen trifft, wenn die Auszahlung (der Nutzen) aus den Aktionen nicht unmittelbar stattfindet, sondern erst nach mehreren Aktionen erfolgt, die *nacheinander* ausgeführt werden. Dieses Thema wurde im Bereich des **Operations Research** verfolgt, der im Zweiten Weltkrieg aus Bemühungen im Vereinigten Königreich zur Optimierung von Radaranlagen entstand und später unzählige zivile Anwendungen fand. Die Arbeit von Richard Bellman (1957) formalisierte eine Klasse von sequenziellen Entscheidungsproblemen, die **Markov-Entscheidungsprobleme**, die wir in Kapitel 17 und, unter der Überschrift des **Reinforcement Learning** (*Verstärkungslernen*), in Kapitel 22 untersuchen.

Die Arbeiten aus den Bereichen Wirtschaftswissenschaften und Operations Research haben viel zu unserem Konzept der rationalen Agenten beigetragen, dennoch entwickelte sich die KI-Forschung viele Jahre lang auf völlig davon getrennten Wegen. Ein Grund dafür war die offensichtliche Komplexität, die mit dem Finden rationaler Entscheidungen einhergeht. Der Pionier der KI-Forschung Herbert Simon (1916–2001) erhielt 1978 den Nobelpreis in Wirtschaftswissenschaften für sein frühes Werk. Darin zeigte er, dass Modelle, die auf **Satisficing**[10] basieren – das heißt, Entscheidungen zu treffen, die „gut genug“ sind, anstatt mühsam eine optimale Entscheidung zu berechnen –, das tatsächliche menschliche Verhalten besser beschreiben (Simon, 1947). Seit den 1990er Jahren ist das Interesse an entscheidungstheoretischen Techniken für die KI wieder gestiegen.

1.2.4 Neurowissenschaft

- Wie verarbeitet das Gehirn Informationen?

Neurowissenschaft ist das Studium des Nervensystems, insbesondere des Gehirns. Obwohl die genaue Art und Weise, wie das Gehirn das Denken ermöglicht, eines der großen Geheim-

10 Zu Deutsch: Zufriedenstellung.

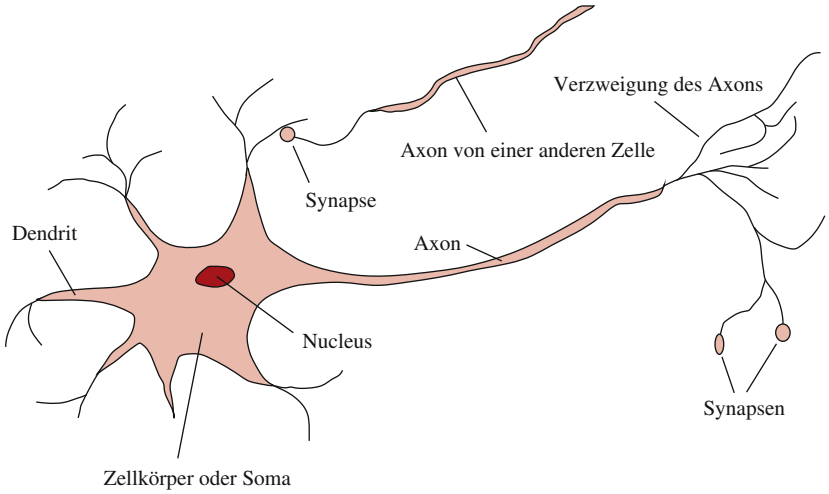


Abbildung 1.1: Bestandteile einer Nervenzelle (Neuron). Jedes Neuron besteht aus einem Zellkörper, Soma, das einen Zellkern enthält. Vom Zellkörper verzweigen sich eine Reihe von Fasern, die Dendriten genannt werden, sowie eine einzelne lange Faser, das Axon. Das Axon erstreckt sich über eine lange Distanz, die größer ist, als der Maßstab in dieser Abbildung andeutet. In der Regel ist ein Axon 1 cm lang (das 100-Fache des Durchmessers des Zellkörpers), es kann aber auch bis zu 1 Meter lang sein. Ein Neuron knüpft Verbindungen mit 10 bis 100.000 anderen Neuronen an Verbindungsstellen, den Synapsen. Signale werden von Neuron zu Neuron mittels einer komplizierten elektrochemischen Reaktion weitergeleitet. Die Signale steuern kurzfristig die Hirnaktivität und ermöglichen auch langfristige Veränderungen in der Verschaltung der Neuronen. Man nimmt an, dass diese Mechanismen die Grundlage für das Lernen im Gehirn bilden. Der größte Teil der Informationsverarbeitung findet in der Großhirnrinde (Kortex) statt, der äußeren Schicht des Gehirns. Die grundlegende organisatorische Einheit scheint eine Art Gewebesäule mit einem Durchmesser von etwa 0,5 mm zu sein, der etwa 20.000 Neuronen enthält und sich über die gesamte Tiefe des Kortex (beim Menschen etwa 4 mm) erstreckt.

nisse der Wissenschaft ist, wird die Tatsache, *dass* es das Denken ermöglicht, seit Tausenden von Jahren anerkannt, denn es ist bewiesen, dass schwere Kopfverletzungen zu geistiger Beeinträchtigung führen können. Es ist auch seit Langem bekannt, dass das menschliche Gehirn irgendwie anders ist; etwa 335 v. Chr. schrieb Aristoteles: „Von allen Tieren hat der Mensch das größte Gehirn im Verhältnis zu seiner Größe.“[11] Dennoch wurde das Gehirn erst Mitte des 18. Jahrhunderts allgemein als Sitz des Bewusstseins anerkannt. Davor kamen unter anderem das Herz und die Milz als Kandidaten infrage.

Mit den Forschungsarbeiten von Paul Broca (1824–1880) zur Aphasie (Sprachstörung) bei hirngeschädigten Patienten im Jahr 1861 begann die Untersuchung der funktionellen Organisation des Gehirns. Broca konnte ein abgegrenztes Areal in der linken Hemisphäre identifizieren – heute Broca-Areal genannt –, das für die Sprachproduktion verantwortlich ist.[12] Zu dieser Zeit war bereits bekannt, dass das Gehirn größtenteils aus Nervenzellen oder **Neuronen** besteht, aber erst 1873 entwickelte Camillo Golgi (1843–1926) eine Färbemethode, die die Beobachtung einzelner Neuronen ermöglichte (▶ Abbildung 1.1). Diese Technik wurde von Santiago Ramón y Cajal (1852–1934) in seinen bahnbrechenden Studien zur neuronalen Organisation verwendet.[13] Es ist heute allgemein anerkannt, dass kognitive Funktionen aus der elektrochemischen Funktionsweise dieser Strukturen resultieren. Das heißt, *eine Ansamm-*

11 Inzwischen hat man herausgefunden, dass das Spitzhörnchen und einige Vogelarten ein größeres Gehirn-Körper-Verhältnis als der Mensch haben.

12 Vielfach wird auch Alexander Hood (1824) als mögliche frühere Quelle zitiert.

13 Golgi beharrte auf seiner Überzeugung, dass die Funktionen des Gehirns in erster Linie in einem kontinuierlichen Medium ausgeführt werden, in das Neuronen eingebettet sind, während Ramón y Cajal die „Neuronendoktrin“ vertrat. Die beiden teilten sich 1906 den Nobelpreis, hielten aber gegenseitig antagonistische Dankesreden.

lung von einfachen Zellen kann zu Denken, Handeln und Bewusstsein führen. In den prägnanten Worten von John Searle (1992), *Gehirn verursacht Geist.*

Heute haben wir einige Daten über die Zuordnungen zwischen Hirnarealen und den Körperteilen, die sie steuern oder von denen sie sensorische Eingaben erhalten. Solche Zuordnungen können sich im Laufe einiger Wochen radikal ändern und einige Tiere scheinen mehrere Zuordnungen zu besitzen. Außerdem verstehen wir nicht vollständig, wie andere Bereiche Funktionen übernehmen können, wenn ein Bereich geschädigt ist. Es gibt fast keine Theorie darüber, wie das Gedächtnis eines Menschen gespeichert wird oder wie kognitive Funktionen auf höherer Ebene arbeiten.

Die Messung der intakten Gehirnaktivität begann 1929 mit der Erfindung der Elektroenzephalografie (EEG) durch Hans Berger. Die Entwicklung der funktionellen Magnetresonanztomografie (fMRT; Ogawa *et al.*, 1990; Cabeza und Nyberg, 2001) liefert den Neurowissenschaftlern beispiellos detaillierte Bilder der Hirnaktivität und ermöglicht Messungen, die auf spannende Weise mit aktuell ablaufenden kognitiven Prozessen korrespondieren. Hinzu kommen Fortschritte in der elektrischen Einzelzellaufzeichnung der Neuronenaktivität und die Methoden der **Optogenetik** (Crick, 1999; Zemelman *et al.*, 2002; Han und Boyden, 2007), die sowohl die Messung als auch die Steuerung einzelner Neuronen ermöglichen, die lichtempfindlich gemacht wurden.

Die Entwicklung von **Gehirn-Maschine-Schnittstellen** (*Brain-Machine-Interface*; Lebedev und Nicolelis, 2006) sowohl für die Sensorik als auch für die motorische Steuerung verspricht nicht nur die Wiederherstellung der Funktionalität von Gliedmaßen bei behinderten Menschen, sondern wirft auch ein Licht auf viele Aspekte neuronaler Systeme. Eine bemerkenswerte Erkenntnis aus dieser Arbeit ist, dass das Gehirn in der Lage ist, sich selbst so einzustellen, dass es erfolgreich mit einem externen Gerät interagieren kann, indem es dieses praktisch wie ein weiteres Sinnesorgan oder ein Körperteil behandelt.

Gehirne und digitale Computer haben recht unterschiedliche Eigenschaften. ► Abbildung 1.2 zeigt, dass die Zykluszeit von Computern eine Million Mal schneller ist als die eines Gehirns. Das Gehirn gleicht dies durch weitaus mehr Speicher und Verbindungen aus, selbst im Vergleich mit einem High-End-PC, obwohl die größten Supercomputer in einigen Punkten mit dem Gehirn gleichziehen. Futuristen geben viel auf diese Zahlen und verweisen auf eine nahende **Singularität**, bei der Computer ein übermenschliches Leistungsniveau erreichen (Vinge, 1993; Kurzweil, 2005; Doctorow und Stross, 2012) und sich dann rapide noch wei-

	Supercomputer	PC	Menschliches Gehirn
Recheneinheiten	10^6 GPUs + CPUs	8 CPU Kerne	10^6 Säulen
	10^{15} Transistoren	10^{10} Transistoren	10^{11} Neuronen
Speichereinheiten	10^{16} Byte RAM	10^{10} Byte RAM	10^{11} Neuronen
	10^{17} Byte Festplatte	10^{12} Byte Festplatte	10^{14} Synapsen
Zykluszeit	10^{-9} s	10^{-9} s	10^{-3} s
Operationen/s	10^{18}	10^{10}	10^{17}

Abbildung 1.2: Ein grober Vergleich zwischen einem führenden Supercomputer, Summit (Feldman, 2017), einem typischen PC im Jahr 2019 und dem menschlichen Gehirn. Menschliche Hirnleistung hat sich in Tausenden von Jahren kaum verändert, Supercomputer haben sich dagegen stetig verbessert: angefangen bei MegaFLOPs in den 1960er Jahren, über GigaFLOPs in den 1980er Jahren, TeraFLOPs in den 1990er Jahren, hin zu PetaFLOPs im Jahr 2008 und ExaFLOPs im Jahr 2018 (1 ExaFLOP = 10^{18} Floating Point Operations per Second / Gleitkommaoperationen pro Sekunde).

ter verbessern. Aber die reinen Zahlenvergleiche sind nicht besonders informativ. Selbst mit einem Computer von nahezu unbegrenzter Kapazität sind wir noch weit davon entfernt, Intelligenz vollständig zu begreifen (siehe Kapitel 28). Grob gesagt: Ohne die richtige Theorie geben schnellere Maschinen einfach schneller die falsche Antwort.

1.2.5 Psychologie

■ Wie denken und handeln Menschen und Tiere?

Die Ursprünge der wissenschaftlichen Psychologie werden gewöhnlich auf die Arbeit des deutschen Physikers Hermann von Helmholtz (1821–1894) und seines Assistenten Wilhelm Wundt (1832–1920) zurückgeführt. Helmholtz wandte die wissenschaftliche Methode auf die Untersuchung der menschlichen Sehkraft an, und sein *Handbuch der Physiologischen Optik* wurde als „die wichtigste Abhandlung über die Physik und Physiologie des menschlichen Sehens“ bezeichnet (Nalwa, 1993, S.15). Im Jahr 1879 eröffnete Wundt das erste Laboratorium für experimentelle Psychologie an der Universität Leipzig. Wundt bestand auf sorgfältig kontrollierten Experimenten, bei denen seine Mitarbeiter eine Wahrnehmungs- oder Assoziationsaufgabe ausführten und dabei ihre Gedankenprozesse selbst beobachteten (Introspektion). Die sorgfältigen Kontrollen trugen viel dazu bei, die Psychologie zu einer Wissenschaft zu erheben, doch die subjektive Natur der Daten machte es unwahrscheinlich, dass die Experimentatoren ihre eigenen Theorien jemals widerlegen würden.

Biologen, die das Verhalten von Tieren untersuchten, verfügten dagegen nicht über introspektive Daten und entwickelten eine objektive Methodik, wie sie von H. S. Jennings (1906) in seinem einflussreichen Werk *Behavior of the Lower Organisms* beschrieben wurde. Die **Behaviorismus**-Bewegung, angeführt von John Watson (1878–1958), übertrug diese Sichtweise auf den Menschen und lehnte *jede* Theorie ab, die mentale Prozesse einbezog, mit der Begründung, dass Introspektion keine zuverlässigen Beweise liefern könne. Die Behavioristen beharrten darauf, nur objektive Messungen der *Reize* (auch *Stimulus* genannt), die auf ein Tier einwirken, sowie die daraus resultierenden Handlungen (*Reaktionen*) zu untersuchen. Der Behaviorismus fand zwar viel über Ratten und Tauben heraus, mit dem Verstehen von Menschen hatte er allerdings weniger Erfolg.

Die **kognitive Psychologie**, die das Gehirn als einen informationsverarbeitenden Apparat betrachtet, lässt sich mindestens bis zu den Arbeiten von William James (1842–1910) zurückführen. Auch Helmholtz vertrat die Ansicht, dass die Wahrnehmung eine Form der unbewussten logischen Schlussfolgerung beinhaltete. Die kognitive Sichtweise stand in den Vereinigten Staaten weitgehend im Schatten des Behaviorismus, doch am Lehrstuhl für Angewandte Psychologie in Cambridge, der von Frederic Bartlett (1886–1969) geleitet wurde, konnte sich die kognitive Modellierung entfalten. Bartletts Schüler und Nachfolger Kenneth Craik (1943) konnte mit seinem Werk *The Nature of Explanation* die Legitimität solcher „mentalen“ Begriffe wie Überzeugungen und Ziele mit Nachdruck wiederherstellen, indem er argumentierte, dass sie genauso wissenschaftlich seien wie beispielsweise die Verwendung von Druck und Temperatur für die Beschreibung von Gas, obwohl Gas aus Molekülen besteht, die weder die eine noch die andere Eigenschaft besitzen.

Craik spezifizierte die drei wichtigsten Schritte eines wissensbasierten Agenten: (1) der Reiz muss in eine interne Repräsentation übersetzt werden, (2) die Repräsentation wird durch kognitive Prozesse manipuliert, um neue interne Repräsentationen abzuleiten, und (3) diese werden wieder zurück in eine Aktion übersetzt. Er erklärte deutlich, warum dies ein gutes Design für einen Agenten ist:

> *Wenn der Organismus ein „kleineres Modell“ der äußeren Realität und seiner eigenen Handlungsmöglichkeiten im Kopf trägt, ist er in der Lage, verschiedene Alternativen auszuprobieren, daraus zu schließen, welche die beste von ihnen ist, auf zukünftige Situationen zu reagieren, bevor sie*

> *entstehen, das Wissen über vergangene Ereignisse im Umgang mit der Gegenwart und der Zukunft zu nutzen und in jeder Hinsicht auf eine viel umfassendere, sicherere und kompetentere Weise auf alle Situationen zu reagieren, mit denen er konfrontiert ist. (Craik, 1943)*

Nach Craiks Tod durch einen Fahrradunfall im Jahr 1945 wurde seine Arbeit von Donald Broadbent fortgesetzt, dessen Buch *Perception and Communication* (1958) eines der ersten Werke war, das psychologische Phänomene als Informationsverarbeitung modellierte. In der Zwischenzeit führte in den Vereinigten Staaten die Entwicklung der Computermodellierung zur Entstehung des Fachbereichs der **Kognitionswissenschaft**. Man sagt, die Kognitionswissenschaft begann auf einem Workshop im September 1956 am MIT – nur zwei Monate nach der Konferenz, auf der die KI selbst „geboren" wurde (siehe S. 40 in Abschnitt 1.3.1).

Auf dem Workshop stellte George Miller *The Magic Number Seven* vor, Noam Chomsky präsentierte *Three Models of Language* und Allen Newell und Herbert Simon stellten *The Logic Theory Machine* vor. Diese drei einflussreichen Arbeiten zeigten, wie Computermodelle verwendet werden können, um die Psychologie des Gedächtnisses, der Sprache und des logischen Denkens zu untersuchen. Es ist heute eine verbreitete (wenn auch bei Weitem nicht universelle) Ansicht unter Psychologen, dass „eine kognitive Theorie wie ein Computerprogramm sein sollte" (Anderson, 1980) – das heißt, sie sollte die Funktionsweise einer kognitiven Funktion in Bezug auf die Verarbeitung von Informationen beschreiben.

Für diesen Rückblick werden wir das Gebiet der **Mensch-Computer-Interaktion** (*Human-Computer Interaction*, HCI) der Psychologie zuordnen. Doug Engelbart, einer der Pioniere der HCI, vertrat die Idee der **erweiterten Intelligenz** (*Intelligence Augmentation*, IA). Er war der Meinung, dass Computer die menschlichen Fähigkeiten erweitern sollten, anstatt menschliche Aufgaben zu automatisieren. Im Jahr 1968 führte Engelbart in einem Vortrag (der später als „The Mother of All Demos" bezeichnet wurde) zum ersten Mal die Computermaus, ein Fenstersystem, Hypertext und Videokonferenzen vor – alles in dem Bemühen zu demonstrieren, was menschliche Wissensträger mit einer Portion erweiterten Intelligenz gemeinsam erreichen könnten.

Heute sehen wir IA und KI eher als zwei Seiten derselben Medaille, wobei erstere die menschliche Kontrolle und letztere das intelligente Verhalten der Maschine betont. Beide sind notwendig, damit Maschinen für den Menschen nutzbringend sind.

1.2.6 Computertechnik

- Wie können wir einen effizienten Computer bauen?

Der moderne digitale elektronische Computer wurde unabhängig und fast zeitgleich von Wissenschaftlern in drei Ländern erfunden, die am Zweiten Weltkrieg beteiligt waren. Der erste *funktionsfähige* Computer war der elektromechanische Heath Robinson[14], der 1943 von Alan Turings Team für einen einzigen Zweck gebaut wurde: die Entschlüsselung deutscher Nachrichten. Die gleiche Gruppe entwickelte 1943 den Colossus, eine leistungsfähige Universalmaschine auf der Basis von Vakuumröhren.[15] Der erste funktionsfähige *programmierbare* Computer war der Z-3, eine Erfindung von Konrad Zuse in Deutschland im Jahr 1941. Zuse erfand außerdem die Gleitkommazahlen und die erste höhere Programmiersprache, Plankalkül. Der erste *elektronische* Computer, der ABC, wurde von John Atanasoff und seinem Doktoranden Clifford Berry zwischen 1940 und 1942 an der Universität von Iowa gebaut. Atanasoffs Forschungen erfuhren wenig Unterstützung oder Anerkennung; es war ENIAC, der im Rah-

14 Eine komplexe Maschine, benannt nach einem britischen Cartoonisten, der skurrile und absurd komplizierte Lösungen für alltägliche Aufgaben wie zum Beispiel das Bestreichen eines Butterbrots zeichnete.

15 In der Zeit nach dem Weltkrieg wollte Turing diese Computer für die KI-Forschung nutzen – er erstellte beispielsweise einen Entwurf des ersten Schachprogramms (Turing *et al.*, 1953) –, doch die britische Regierung blockierte diese Forschung.

men eines geheimen Militärprojekts an der Universität von Pennsylvania von einem Team um John Mauchly und J. Presper Eckert entwickelt wurde und der sich als der einflussreichste Vorläufer der modernen Computer erwies.

Seitdem hat jede Generation von Computerhardware eine Steigerung hinsichtlich Geschwindigkeit und Kapazität sowie einen Rückgang hinsichtlich des Preises mit sich gebracht – ein Trend, der im **Moore'schen Gesetz** festgehalten ist. Bis etwa ins Jahr 2005 verdoppelte sich die Leistung etwa alle 18 Monate. Ab dann begannen die Hersteller aufgrund von Problemen mit der Verlustleistung die Anzahl der CPU-Kerne anstatt die Taktrate zu erhöhen. Zur Zeit geht man davon aus, dass künftige Steigerungen der Funktionalität durch massive Parallelität erreicht werden – eine merkwürdige Übereinstimmung mit den Eigenschaften des Gehirns. Wir erleben auch neue Hardwaredesigns, die auf der Idee basieren, dass wir im Umgang mit einer unsicheren Welt keine 64-Bit-Präzision in unseren Zahlen benötigen; 16 Bit (wie im `bfloat16`-Format)[16] oder sogar 8 Bit reichen aus und ermöglichen eine schnellere Verarbeitung.

Wir stehen gerade am Anfang einer Hardwareentwicklung, die auf KI-Anwendungen abgestimmt ist, wie z. B. Grafikprozessoren (*Graphics Processing Unit*, GPU), Tensorprozessoren (*Tensor Processing Unit*, TPU) oder die Wafer Scale Engine (WSE). Von den 1960er Jahren bis etwa ins Jahr 2012 folgte die Höhe der Rechenleistung, die zum Trainieren der Top-Anwendungen für das maschinelle Lernen verwendet wurde, dem Moore'schen Gesetz. Ab 2012 änderte sich die Lage: Von 2012 bis 2018 gab es einen 300.000-fachen Anstieg, was ungefähr einer Verdoppelung alle 100 Tage entspricht (Amodei und Hernandez, 2018). Ein Modell des maschinellen Lernens, das 2014 einen ganzen Tag zum Trainieren benötigte, brauchte 2018 nur noch zwei Minuten (Ying *et al.*, 2018). Obwohl aktuell noch nicht praxistauglich, versprechen **Quantencomputer** noch weitaus größere Beschleunigungen für einige wichtige Unterklassen von KI-Algorithmen.

Natürlich gab es auch vor dem elektronischen Computer schon Rechengeräte. Die frühesten automatisierten Maschinen aus dem 17. Jahrhundert wurden bereits auf S. 27 angesprochen. Die erste *programmierbare* Maschine war ein Webstuhl, der 1805 von Joseph Marie Jacquard (1752–1834) entwickelt wurde und der Anweisungen für das zu webende Muster mithilfe von Lochkarten speicherte.

Mitte des 19. Jahrhunderts entwarf Charles Babbage (1792–1871) zwei Rechenmaschinen, die er aber beide nicht fertigstellte. Seine Differenzmaschine („Difference Engine") sollte mathematische Tabellen für technische und wissenschaftliche Projekte berechnen. Sie wurde schließlich zwischen 1989 und 1991 nachgebaut und erwies sich als funktionstüchtig (Swade, 2000). Babbages „Analytical Engine" war weitaus ambitionierter: Sie verfügte über einen adressierbaren Speicher, gespeicherte Programme, die auf Jacquards Lochkarten basierten, sowie über bedingte Sprünge. Sie war die erste Maschine, die universelle Berechnungen durchführen konnte.

Babbages Kollegin Ada Lovelace, Tochter des Dichters Lord Byron, verstand das Potenzial der Analytical Engine und beschrieb sie als „eine denkende oder … eine schlussfolgernde Maschine", die in der Lage sei, über „alle Themen im Universum" zu räsonieren (Lovelace, 1843). Sie nahm auch den Hype-Zyklus der KI vorweg, indem sie schrieb: „Es ist wünschenswert, sich vor der Möglichkeit übertriebener Ideen zu schützen, die hinsichtlich der Leistungen der Analytical Engine entstehen könnten." Leider gerieten Babbages Maschinen und Lovelaces Ideen weitgehend in Vergessenheit.

Die KI ist auch der Softwareseite der Informatik sehr zu Dank verpflichtet, von der Betriebssysteme, Programmiersprachen und Werkzeuge bereitgestellt wurden, die zur Entwicklung moderner Programme (und der entsprechenden Veröffentlichung dazu) nötig sind. Doch hier

16 *Brain floating point with 16 bits.*

wurde die Schuld beglichen: Die Tätigkeiten im Bereich der KI haben viele Ideen hervorgebracht, die ihren Weg zurück in die Mainstream-Informatik gefunden haben, darunter Time-Sharing, interaktive Interpreter, PCs mit Fenstern und Mäusen, schnelle Entwicklungsumgebungen, der Datentyp der verknüpften Liste, automatische Speicherverwaltung und Schlüsselkonzepte der symbolischen, funktionalen, deklarativen und objektorientierten Programmierung.

1.2.7 Kontrolltheorie und Kybernetik

■ Wie können Artefakte unter eigener Steuerung arbeiten?

Ktesibios von Alexandria (ca. 250 v. Chr.) baute die erste selbststeuernde Maschine: eine Wasseruhr mit einem Regler, der eine konstante Durchflussrate aufrechterhielt. Diese Erfindung veränderte die Definition dessen, was ein Artefakt tun konnte. Bis dahin konnten nur Lebewesen ihr Verhalten als Reaktion auf Veränderungen in der Umwelt anpassen. Andere Beispiele für selbstregulierende rückgekoppelte Steuerungssysteme sind u. a. der Dampfmaschinenregler von James Watt (1736–1819) und der Thermostat von Cornelis Drebbel (1572–1633), der auch das U-Boot erfand. James Clerk Maxwell (1868) begründete die mathematische Theorie der Regelsysteme.

Eine zentrale Figur der Nachkriegszeit bei der Entwicklung der **Kontrolltheorie** war Norbert Wiener (1894–1964). Wiener war ein brillanter Mathematiker, der u. a. mit Bertrand Russell zusammenarbeitete, bevor er anfing, sich für biologische und mechanische Kontrollsysteme und deren Verbindung zur Kognition zu interessieren. Wie Craik (der ebenfalls Kontrollsysteme als psychologische Modelle verwendete) stellten Wiener und seine Kollegen Arturo Rosenblueth und Julian Bigelow die behavioristische Orthodoxie infrage (Rosenblueth *et al.*, 1943). Sie betrachteten zweckgerichtetes Handeln als Folge eines Regulationsmechanismus, durch den versucht wird, „Fehler" – die Differenz zwischen dem aktuellen Zustand und dem Zielzustand – zu minimieren. Ende der 1940er Jahre organisierte Wiener zusammen mit Warren McCulloch, Walter Pitts und John von Neumann eine Reihe einflussreicher Konferenzen, die die neuen mathematischen und computergestützten Modelle der Kognition untersuchten. Wieners Buch *Cybernetics* (1948) wurde ein Bestseller und machte die Öffentlichkeit auf die Möglichkeit aufmerksam, Maschinen mit künstlicher Intelligenz auszustatten.

In der Zwischenzeit bereitete W. Ross Ashby im Vereinigten Königreich den Weg für ähnliche Ideen (Ashby, 1940). Ashby, Alan Turing, Grey Walter und andere gründeten den Ratio Club für „diejenigen, die Wieners Ideen hatten, bevor Wieners Buch erschien". In *Design for a Brain* (1948, 1952) führte Ashby seine Idee weiter aus, dass Intelligenz mithilfe **homöostatischer** Geräte erzeugt werden kann, in denen mit geeigneten Rückkopplungsschleifen ein stabiles adaptives Verhalten erreicht wird.

Die moderne Kontrolltheorie, insbesondere der als stochastische optimale Steuerung bekannte Zweig, hat das Ziel, Systeme zu entwerfen, die eine **Kostenfunktion** über die Zeit minimieren. Dies entspricht in etwa dem Standardmodell der KI: Systeme zu entwerfen, die sich optimal verhalten. Warum also sind KI und Kontrolltheorie zwei unterschiedliche Gebiete, trotz der engen Verbindungen zwischen ihren Begründern? Die Antwort liegt in der engen Kopplung zwischen den mathematischen Techniken, die den Beteiligten vertraut waren, und den entsprechenden Problemstellungen, die in jeder Weltsicht enthalten waren. Analysis und Matrizenalgebra, die Werkzeuge der Kontrolltheorie, eignen sich für Systeme, die durch feste Mengen kontinuierlicher Variablen beschreibbar sind, während KI zum Teil als eine Möglichkeit gegründet wurde, diesen wahrgenommenen Einschränkungen zu entkommen. Die Werkzeuge der logischen Inferenz und der Berechnung erlaubten es den KI-Forschern, Probleme wie Sprache, Sehen und symbolische Planung zu betrachten, die völlig außerhalb des Aufgabenbereichs der Kontrolltheoretiker lagen.

1.2.8 Linguistik

- Wie hängen Sprache und Denken zusammen?

1957 veröffentlichte B. F. Skinner sein Buch *Verbal Behavior* – eine umfassende, detaillierte Darstellung des behavioristischen Ansatzes zum Spracherwerb, geschrieben von dem führenden Experten auf diesem Gebiet. Doch seltsamerweise wurde eine Rezension des Buchs so bekannt wie das Buch selbst und brachte das Interesse am Behaviorismus fast zum Erliegen. Der Autor der Kritik war der Linguist Noam Chomsky, der gerade ein Buch über seine eigene Theorie, *Syntactic Structures*, veröffentlicht hatte. Chomsky wies darauf hin, dass die behavioristische Theorie nicht auf den Begriff der Kreativität in der Sprache einging – sie erklärte nicht, wie Kinder Sätze verstehen und sich ausdenken konnten, die sie noch nie zuvor gehört hatten. Chomskys Theorie – basierend auf syntaktischen Modellen, die auf den indischen Linguisten Panini (ca. 350 v. Chr.) zurückgehen – konnte dies erklären und im Gegensatz zu früheren Theorien war sie formal genug, dass sie im Prinzip programmiert werden konnte.

Die moderne Linguistik und die KI wurden also etwa zur gleichen Zeit „geboren“ und wuchsen zusammen auf, mit einer Schnittmenge in einem hybriden Bereich, der **Computerlinguistik** oder **natürlichen Sprachverarbeitung**. Das Problem, Sprache zu verstehen, stellte sich als wesentlich komplexer heraus, als es 1957 den Anschein hatte. Das Verstehen von Sprache erfordert ein Begreifen des Themas und des Kontexts, nicht nur der Struktur von Sätzen. Dies mag offensichtlich erscheinen, war aber bis in die 1960er Jahre nicht allgemein anerkannt. Ein Großteil der frühen Arbeiten im Bereich der **Wissensrepräsentation** (das Studium darüber, wie Wissen in eine Form gebracht wird, mit der ein Computer umgehen kann) war an Sprache gebunden und wurde durch die linguistische Forschung unterstützt, die wiederum durch jahrzehntelange Arbeit mit der philosophischen Analyse von Sprache verbunden war.

1.3 Die Geschichte der Künstlichen Intelligenz

Eine schnelle Möglichkeit, einen Überblick über die Meilensteine der KI-Geschichte zu bekommen, ist die Auflistung der Preisträger des Turing Awards: Marvin Minsky (1969) und John McCarthy (1971) für die Definition der Grundlagen des Forschungsgebiets auf der Basis von Repräsentation und Schlussfolgerungen; Allen Newell und Herbert Simon (1975) für symbolische Modelle der Problemlösung und menschlicher Kognition; Ed Feigenbaum und Raj Reddy (1994) für die Entwicklung von Expertensystemen, die menschliches Wissen codieren, um reale Probleme zu lösen; Judea Pearl (2011) für die Entwicklung probabilistischer Schlussfolgerungstechniken, die mit Unsicherheiten auf grundlegende Weise umgehen; und schließlich Yoshua Bengio, Geoffrey Hinton und Yann LeCun (2019), die „Deep Learning“ (mehrschichtige neuronale Netze) zu einem wichtigen Bestandteil der modernen Datenverarbeitung gemacht haben. Im weiteren Verlauf dieses Abschnitts gehen wir näher auf die einzelnen Phasen der KI-Geschichte ein.

1.3.1 Die Anfänge der Künstlichen Intelligenz (1943–1956)

Die erste Arbeit, die heute allgemein als KI anerkannt wird, stammt von Warren McCulloch und Walter Pitts (1943). Inspiriert von der mathematischen Modellierungsarbeit von Pitts' Berater Nicolas Rashevsky (1936, 1938), basiert sie auf drei Quellen: dem Wissen über die grundlegende Physiologie und die Funktion von Neuronen im Gehirn; einer formalen Analyse der Aussagenlogik, die auf Russell und Whitehead zurückgeht; sowie Turings Theorie der Berechenbarkeit. Sie schlugen ein Modell künstlicher Neuronen vor, in dem jedes Neuron als „an“ oder „aus“ charakterisiert wird, wobei ein Wechsel auf „an“ als Reaktion auf die Aktivierung durch eine ausreichende Anzahl benachbarter Neuronen erfolgt. Der Zustand eines Neurons wurde aufgefasst als „faktisch äquivalent zu einer logischen Aussage, die ihren

adäquaten Reiz darstellt". Sie zeigten zum Beispiel, dass jede berechenbare Funktion durch ein Netzwerk verbundener Neuronen berechnet werden kann und dass alle logischen Verknüpfungen (UND, ODER, NICHT usw.) durch einfache Netzstrukturen implementiert werden können. McCulloch und Pitts behaupteten außerdem, dass geeignet definierte Netze lernfähig seien. Donald Hebb (1949) demonstrierte eine einfache Aktualisierungsregel, mit deren Hilfe die Verbindungsstärken zwischen Neuronen verändert werden können. Seine Regel, heute als **Hebb'sches Lernen** bezeichnet, ist immer noch ein einflussreiches Modell.

Zwei Studenten in Harvard, Marvin Minsky (1927–2016) und Dean Edmonds, bauten 1950 den ersten neuronalen Netzcomputer – SNARC. SNARC verwendete 3.000 Vakuumröhren und einen nicht mehr benötigten Autopilotmechanismus aus einem B-24-Bomber, um ein Netzwerk aus 40 Neuronen zu simulieren. Minsky studierte später in Princeton universelle Berechnungen in neuronalen Netzwerken. Sein Promotionsausschuss war skeptisch, ob diese Art von Arbeit als Mathematik angesehen werden konnte, doch von Neumann soll gesagt haben: „Wenn es heute noch nicht so ist, dann wird es eines Tages so sein."

Es gab eine Reihe weiterer Beispiele für frühe Arbeiten, die als KI charakterisiert werden können, darunter zwei Dameprogramme, die 1952 unabhängig voneinander von Christopher Strachey an der Universität von Manchester und von Arthur Samuel bei IBM entwickelt wurden. Am einflussreichsten war jedoch die Vision von Alan Turing. Bereits 1947 hielt er bei der London Mathematical Society Vorträge zu diesem Thema und formulierte 1950 in seinem Artikel „Computing Machinery and Intelligence" eine überzeugende Agenda. Darin stellte er den Turing-Test, maschinelles Lernen, genetische Algorithmen und Reinforcement Learning vor. Er ging auf viele der Einwände ein, die gegen die Möglichkeit von KI erhoben wurden, was wir in Kapitel 27 noch ausführen werden. Außerdem vermutete er, eine KI auf menschlichem Niveau zu schaffen wäre einfacher, wenn man Lernalgorithmen entwickelt und mit diesen dann die Maschine trainiert, anstatt ihre Intelligenz von Hand zu programmieren. In späteren Vorträgen warnte er, dass das Erreichen dieses Ziels vielleicht nicht vorteilhaft für die Menschheit wäre.

Im Jahr 1955 überzeugte John McCarthy vom Dartmouth College seine Kollegen Marvin Minsky, Claude Shannon und Nathaniel Rochester, ihm dabei zu helfen, Forscher aus den USA zusammenzubringen, die an der Automatentheorie, an neuronalen Netzen und der Untersuchung von Intelligenz interessiert waren. Sie organisierten im Sommer 1956 einen zweimonatigen Workshop in Dartmouth. Insgesamt nahmen 10 Personen daran teil, darunter Allen Newell und Herbert Simon von der Carnegie Tech[17], Trenchard More aus Princeton, Arthur Samuel von IBM sowie Ray Solomonoff und Oliver Selfridge vom MIT. Im Antrag heißt es:[18]

> *Wir schlagen vor, im Sommer 1956 am Dartmouth College in Hanover, New Hampshire, eine zweimonatige Studie mit 10 Personen zur künstlichen Intelligenz durchzuführen. Die Studie soll aufgrund der Vermutung stattfinden, dass jeder Aspekt des Lernens oder jedes andere Merkmal der Intelligenz im Prinzip so genau beschrieben werden kann, dass eine Maschine gebaut werden kann, die dies simuliert. Es soll versucht werden herauszufinden, wie man Maschinen dazu bringt, Sprache zu benutzen, Abstraktionen und Konzepte zu bilden, Probleme zu lösen, die heute dem Menschen vorbehalten sind, und sich selbst zu verbessern. Wir glauben, dass ein signifikanter Fortschritt in einem oder mehreren dieser Probleme erzielt werden kann, wenn eine sorgfältig ausgewählte Gruppe von Wissenschaftlern einen Sommer lang gemeinsam daran arbeitet.*

17 Heute Carnegie Mellon University (CMU).

18 Dies war die erste offizielle Verwendung von McCarthys Begriff der *künstlichen Intelligenz*. Vielleicht wäre „berechenbare Rationalität" präziser und weniger bedrohlich gewesen, doch „KI" ist geblieben. Anlässlich des 50. Jahrestags der Dartmouth-Konferenz erklärte McCarthy, dass er die Begriffe „Computer" oder „berechenbar" aus Rücksicht auf Norbert Wiener vermieden hatte, der sich mit analogen kybernetischen Geräten und nicht mit digitalen Computern beschäftigte.

Dieser optimistischen Vorhersage zum Trotz führte der Workshop in Dartmouth zu keinem Durchbruch. Newell und Simon präsentierten die vielleicht am weitesten ausgereifte Arbeit, einen mathematischen Theorembeweiser namens Logic Theorist (LT). Simon behauptete: „Wir haben ein Computerprogramm entworfen, das in der Lage ist, nichtnumerisch zu denken, und haben damit das altehrwürdige Körper-Geist-Problem gelöst.“[19] Kurz nach dem Workshop war das Programm in der Lage, die meisten der Theoreme in Kapitel 2 der *Principia Mathematica* von Russell und Whitehead zu beweisen. Russell soll erfreut gewesen sein, als er erfuhr, dass der LT einen Beweis für ein Theorem gefunden hatte, der kürzer als der in der *Principia* war. Die Redakteure des *Journal of Symbolic Logic* waren weniger beeindruckt – sie lehnten einen Aufsatz ab, der von Newell, Simon und dem Logic Theorist verfasst wurde.

1.3.2 Früher Enthusiasmus, große Erwartungen (1952–1969)

Das intellektuelle Establishment der 1950er Jahre zog es im Großen und Ganzen vor zu glauben, dass „eine Maschine niemals *X* tun kann“. (In Kapitel 27 finden Sie eine lange Liste der von Turing zusammengetragenen *X*e.) Die Antwort der KI-Forscher bestand natürlich darin, ein *X* nach dem anderen vorzuführen. Dabei konzentrierten sie sich insbesondere auf Aufgaben, die als Indikator für die Intelligenz von Menschen gelten, zum Beispiel Spiele, Rätsel, Mathematik und IQ-Tests. John McCarthy bezeichnete diese Periode als die „Look, Ma, no hands!“-Ära („Mama, guck mal, freihändig!“).

Newell und Simon knüpften an ihren LT-Erfolg mit dem General Problem Solver, kurz GPS, an. Anders als der LT war dieses Programm von Anfang an darauf ausgelegt, menschliche Protokolle zur Problemlösung zu imitieren. Innerhalb der begrenzten Klasse logischer Rätsel, die er bearbeiten konnte, stellte sich heraus, dass die Reihenfolge, in der das Programm Teilziele und mögliche Aktionen in Betracht zog, der Reihenfolge ähnelte, in der Menschen dieselbe Art Probleme angehen. Damit war der GPS wahrscheinlich das erste Programm, das den Ansatz des „menschlichen Denkens“ verkörperte. Der Erfolg von GPS und nachfolgenden Programmen als Modelle der Kognition veranlasste Newell und Simon (1976), ihre berühmte Hypothese des **physischen Symbolsystems** zu formulieren, die besagt, dass „ein physisches Symbolsystem die notwendigen und hinreichenden Mittel für allgemein intelligentes Handeln besitzt“. Sie meinten damit, dass jedes System (Mensch oder Maschine), das Intelligenz zeigt, durch die Manipulation von Datenstrukturen, die aus Symbolen bestehen, funktionieren muss. Wir werden später noch sehen, dass diese Hypothese aus vielen Richtungen infrage gestellt wurde.

Bei IBM erstellten Nathaniel Rochester und seine Kollegen einige der ersten KI-Programme. Herbert Gelernter (1959) konstruierte den Geometry Theorem Prover, der in der Lage war, Theoreme zu beweisen, die viele Mathematikstudenten als zu knifflig empfunden hätten. Diese Arbeit war eine Vorstufe der modernen mathematischen Theorembeweiser.

Von allen Forschungsarbeiten dieser Zeit war die von Arthur Samuel zum Damespiel vermutlich langfristig die einflussreichste. Unter Verwendung von Methoden, die wir heute als Reinforcement Learning bezeichnen (siehe Kapitel 22), lernten Samuels Programme, auf hohem Amateurniveau zu spielen. Damit widerlegte er die Vorstellung, dass Computer nur das tun können, was man ihnen sagt: Sein Programm lernte schnell, besser zu spielen als sein Schöpfer. Das Programm wurde 1956 im Fernsehen vorgeführt und hinterließ einen starken Eindruck. Wie Turing hatte auch Samuel Schwierigkeiten, Rechenzeit zu bekommen. Er arbeitete nachts und benutzte Maschinen, die sich bei IBM noch in der Testphase befanden. Samuels Programm war der Vorläufer späterer Systeme wie TD-GAMMON (Tesauro, 1992), das

19 Um den LT zu schreiben, erfanden Newell und Simon außerdem eine listenverarbeitende Sprache, IPL. Sie hatten keinen Compiler und übersetzten sie von Hand in Maschinencode. Um Fehler zu vermeiden, arbeiteten sie parallel und riefen sich beim Schreiben jeder Anweisung gegenseitig Binärzahlen zu, um sicherzustellen, dass sie übereinstimmten.

zu den weltbesten Backgammon-Spielern gehörte, und ALPHAGO (Silver *et al.*, 2016), das die Welt schockierte, als es den menschlichen Weltmeister im Go besiegte (siehe Kapitel 5).

Im Jahr 1958 leistete John McCarthy zwei wichtige Beiträge zur KI. Im „MIT AI ab Memo No. 1“ definierte er die höhere Programmiersprache **Lisp**, die für die nächsten 30 Jahre die dominierende KI-Programmiersprache werden sollte. In einem Aufsatz mit dem Titel *Programs with Common Sense* („Programme mit gesundem Menschenverstand“) unterbreitete er einen konzeptionellen Vorschlag für KI-Systeme, die auf Wissen und Schlussfolgerungen basieren. McCarthy beschreibt in dieser Arbeit den Advice Taker, ein hypothetisches Programm, das Allgemeinwissen über die Welt verkörpern würde und daraus Handlungspläne ableiten könnte. Das Konzept wurde mit einfachen logischen Axiomen veranschaulicht, die ausreichen, um einen Plan für die Fahrt zum Flughafen zu generieren. Das Programm war außerdem so konzipiert, dass es im normalen Betrieb neue Axiome akzeptieren und so Kompetenz in neuen Bereichen erlangen konnte, *ohne neu programmiert zu werden*. Der Advice Taker verkörperte damit die zentralen Prinzipien der Wissensrepräsentation und des Schlussfolgerns: dass es sinnvoll ist, eine formale, explizite Repräsentation der Welt und ihrer Funktionsweise zu haben sowie in der Lage zu sein, diese Repräsentation mit deduktiven Verfahren zu verändern. Der Artikel beeinflusste den Verlauf der KI und ist bis heute relevant.

1958 war auch das Jahr, in dem Marvin Minsky ans MIT wechselte. Seine anfängliche Zusammenarbeit mit McCarthy war jedoch nicht von Dauer. McCarthy konzentrierte sich auf Repräsentation und Schlussfolgern in formaler Logik, Minsky hingegen war mehr daran interessiert, Programme zum Laufen zu bringen – er entwickelte schließlich eine antilogische Anschauung. Im Jahr 1963 gründete McCarthy das AI Lab in Stanford. Sein Plan, mithilfe von Logik den ultimativen Advice Taker zu bauen, wurde durch J. A. Robinsons Entdeckung des Resolutionsverfahrens (ein vollständiger Algorithmus zum Beweisen von Theoremen der Prädikatenlogik, siehe Kapitel 9) im Jahr 1965 vorangetrieben. Die Arbeiten in Stanford betonten allgemeine Methoden für logisches Schlussfolgern. Zu den Anwendungen der Logik gehörten Cordell Greens Frage-Antwort-Systeme und Planungssysteme (Green, 1969b) und das Shakey-Robotikprojekt am Stanford Research Institute (SRI). Das Shakey-Projekt, auf das wir in Kapitel 26 näher eingehen, war das erste, das die vollständige Integration von logischem Schlussfolgern und physischer Aktivität demonstrierte.

Am MIT betreute Minsky eine Reihe von Studenten, die Aufgaben aus einem begrenzten Problembereich auswählten, deren Lösung Intelligenz zu erfordern schien. Diese begrenzten Domänen wurden unter dem Namen **Mikrowelten** bekannt. Das Programm SAINT von James Slagle (1963) war in der Lage, geschlossene Integrationsprobleme zu lösen, die typischerweise in Kursen des ersten Grundstudiumsjahrs vorkommen. Das Programm ANALOGY von Tom Evans (1968) löste geometrische Analogieprobleme, die in IQ-Tests vorkommen. STUDENT von Daniel Bobrow (1967) löste Algebra-Textaufgaben, wie z. B. die folgende:

> *Wenn die Anzahl der Kunden, die Tom hat, doppelt so groß ist wie das Quadrat von 20 Prozent der Anzeigen, die Tom aufgegeben hat, und die Anzahl der Anzeigen 45 beträgt – wie viele Kunden hat Tom dann?*

Die bekannteste Mikrowelt ist die **Blockwelt**, die aus einer Reihe von festen Blöcken besteht, die auf einer Tischplatte (oder häufiger auf der Simulation einer Tischplatte) platziert sind, wie in ► Abbildung 1.3 gezeigt. Eine typische Aufgabe in dieser Welt besteht darin, die Blöcke in einer bestimmten Weise umzuordnen, wozu eine Roboterhand verwendet wird, die jeweils einen Block aufheben kann. Die Blockwelt war Ausgangspunkt des Vision-Projekts von David Huffman (1971), der Visions- und Constraint-Propagation-Arbeit von David Waltz (1975), der Theorie des Lernens von Patrick Winston (1970), des Programms zum Verstehen natürlicher Sprache von Terry Winograd (1972) und des Planers von Scott Fahlman (1974).

Frühe Arbeiten, die auf den neuronalen Netzen von McCulloch und Pitts aufbauten, erlebten ebenfalls eine Blütezeit. Die Arbeit von Shmuel Winograd und Jack Cowan (1963) zeigte,

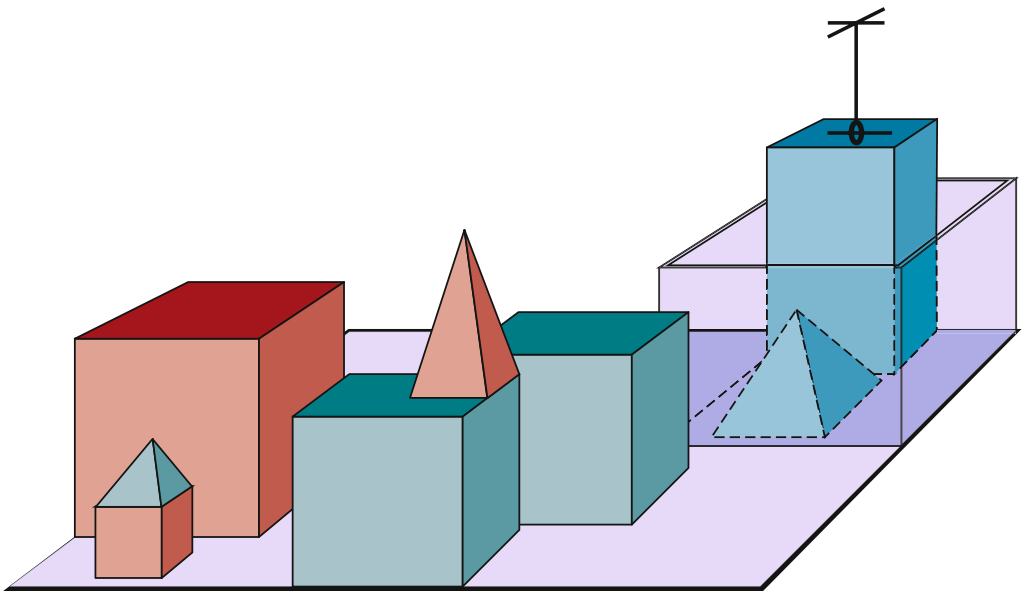


Abbildung 1.3: Eine Szene aus der Blockwelt. SHRDLU (Winograd, 1972) hat soeben den Befehl „Finde einen Block, der größer ist als derjenige, den du gerade hältst, und lege ihn in den Kasten" verarbeitet.

wie eine große Anzahl von Elementen kollektiv ein individuelles Konzept repräsentieren konnten, mit einer entsprechenden Steigerung an Robustheit und Parallelität. Hebbs Lernmethoden wurden von Bernie Widrow (Widrow und Hoff, 1960; Widrow, 1962), der seine Netze als **Adalines** bezeichnete, und von Frank Rosenblatt (1962) mit seinem **Perzeptron**-Konzept weiterentwickelt. Das **Perzeptron-Konvergenz-Theorem** (Block *et al.*, 1962) besagt, dass der Lernalgorithmus die Verbindungsstärken eines Perzeptrons an beliebige Eingabedaten anpassen kann, sofern eine solche Anpassung existiert.

1.3.3 Eine Portion Realität (1966–1973)

Die KI-Forscher scheuten sich von Anfang an nicht, ihre kommenden Erfolge vorherzusagen. Die folgende Aussage von Herbert Simon aus dem Jahr 1957 wird oft zitiert:

> *Es liegt mir nicht daran, Sie zu überraschen oder zu schockieren – doch um es möglichst einfach und kurz auszudrücken: Es gibt heute in der Welt Maschinen, die denken, die lernen und die kreativ sind. Darüber hinaus wird ihre Fähigkeit, dies zu tun, schnell zunehmen, bis – in absehbarer Zukunft – die Bandbreite der Probleme, die sie bewältigen können, genauso groß sein wird wie die Bereiche, die der menschliche Verstand bewältigen kann.*

Der Ausdruck „absehbare Zukunft" ist zwar vage, aber Simon machte auch konkretere Vorhersagen: dass innerhalb von 10 Jahren ein Computer Schachweltmeister sein und ein bedeutendes mathematisches Theorem maschinell bewiesen sein würde. Diese Vorhersagen bewahrheiteten sich tatsächlich (zumindest annähernd), allerdings innerhalb von 40 Jahren und nicht von 10 Jahren. Simons übersteigerte Zuversicht war auf die vielversprechende Leistung früher KI-Systeme für einfache Beispiele zurückzuführen. In fast allen Fällen versagten diese frühen Systeme jedoch bei schwierigeren Problemen.

Für dieses Scheitern gab es zwei Hauptgründe. Der erste war, dass viele frühe KI-Systeme in erster Linie auf „informierter Introspektion" basierten, wie Menschen eine Aufgabe ausführen, anstatt darauf, die Aufgabe sorgfältig zu analysieren; herauszufinden, wie die Lösungen

beschaffen sein muss; und was ein Algorithmus tun müsste, um solche Lösungen zuverlässig zu produzieren.

Der zweite Grund für das Scheitern war das mangelnde Bewusstsein für das Komplexitätsverhalten der Probleme, die die KI zu lösen versuchte, das heißt, wie sich die Laufzeit mit der Größe der Eingabemenge ändert (die *effiziente* Machbarkeit). In den meisten der frühen Systeme zum automatischen Problemlösen wurden verschiedene Schrittkombinationen ausprobiert, bis die Lösung gefunden war. Diese Strategie funktionierte anfangs, weil Mikrowelten nur sehr wenige Objekte und damit nur wenige mögliche Aktionen und sehr kurze Lösungsfolgen enthielten. Bevor die Komplexitätstheorie entwickelt wurde, war man weithin der Meinung, dass die „Skalierung" auf größere Probleme einfach eine Frage von schnellerer Hardware und größerem Speicher sei. Der Optimismus, der zum Beispiel die Entwicklung des Theorembeweisens mittels Resolution begleitete, wurde bald gedämpft, als es den Forschern nicht gelang, Theoreme mit mehr als ein paar Dutzend Fakten zu beweisen. *Die Tatsache, dass ein Programm prinzipiell eine Lösung finden kann, bedeutet nicht, dass das Programm irgendeinen Mechanismus enthält, der nötig ist, um diese Lösung auch in der Praxis zu finden.*

Die Illusion unbegrenzter Rechenleistung war nicht auf Problemlösungsprogramme beschränkt. Frühe Experimente in der **Maschinenevolution** (heute als **genetische Algorithmen** bezeichnet; Friedberg, 1958; Friedberg *et al.*, 1959) basierten auf der zweifellos richtigen Überzeugung, dass man durch eine geeignete Reihe kleiner Mutationen im Maschinencode ein effizientes Programm für jede konkrete Aufgabe erzeugen kann. Die Idee war, zufällige Mutationen auszuprobieren und dabei einen Selektionsprozess zu verwenden, der nützlich erscheinende Mutationen beibehielt. Trotz Tausender von Stunden CPU-Zeit konnte fast kein Fortschritt nachgewiesen werden.

Das Versäumnis, die „kombinatorische Explosion" in den Griff zu bekommen, war einer der Hauptkritikpunkte an der KI im Lighthill-Bericht (Lighthill, 1973), auf dessen Grundlage die britische Regierung entschied, die Unterstützung der KI-Forschung in allen außer zwei Universitäten einzustellen. (Die mündliche Überlieferung zeichnet ein etwas anderes und bunteres Bild, mit politischen Ambitionen und persönlichen Animositäten, die hier nicht weiter ausgeführt werden sollen.)

Eine dritte Schwierigkeit ergab sich aus einigen fundamentalen Einschränkungen der grundlegenden Strukturen, die zur Erzeugung intelligenten Verhaltens verwendet werden. Zum Beispiel bewiesen Minsky und Papert in ihrem Buch *Perceptrons* (1969), dass Perzeptronen (eine einfache Form eines neuronalen Netzes) zwar nachweislich alles lernen können, was sie auch darstellen können, aber sie können nur recht wenig darstellen. Insbesondere konnte ein Perzeptron mit zwei Eingängen nicht darauf trainiert werden zu erkennen, wann die Werte an den Eingängen unterschiedlich waren. Obwohl ihre Ergebnisse nicht auf komplexere, mehrschichtige Netzwerke zutrafen, wurden die Forschungsgelder für neuronale Netze bald fast auf null zurückgefahren. Ironischerweise waren ausgerechnet die neuen Rückwärtspropagation-Lernalgorithmen, die in den späten 1980er Jahren und dann noch einmal in den 2010er Jahren für eine enorme Wiederbelebung der Neuronale-Netze-Forschung sorgen sollten, bereits in den frühen 1960er Jahren in anderen Zusammenhängen entwickelt worden (Kelley, 1960; Bryson, 1962).

1.3.4 Expertensysteme (1969–1986)

Das Bild des Problemlösens, das sich im ersten Jahrzehnt der KI-Forschung herausgebildet hatte, war das eines universellen Suchmechanismus, der versucht, elementare Schlussfolgerungsschritte aneinanderzureihen, um vollständige Lösungen zu finden. Solche Ansätze wurden als **schwache Methoden** bezeichnet, weil sie zwar allgemein sind, aber nicht auf große oder schwierige Probleminstanzen skaliert werden können. Die Alternative zu schwachen

Methoden besteht darin, leistungsfähigeres, fachbereichspezifisches Wissen zu verwenden, das größere Schlussfolgerungsschritte erlaubt und Fälle, die typischerweise in eingeschränkteren Fachgebieten auftreten, leichter behandeln kann. Man könnte sagen, dass man, um ein schweres Problem zu lösen, die Antwort fast schon kennen muss.

Das Programm DENDRAL (Buchanan *et al.*, 1969) war ein frühes Beispiel für diesen Ansatz. Es wurde in Stanford entwickelt, wo sich Ed Feigenbaum (ein ehemaliger Student von Herbert Simon), Bruce Buchanan (ein Philosoph, der zum Informatiker wurde) und Joshua Lederberg (ein Genetiker und Nobelpreisträger) zusammenschlossen, um das Problem der Ableitung molekularer Strukturen aus den Informationen eines Massenspektrometers zu lösen. Die Eingabe für das Programm besteht aus der elementaren Formel des Moleküls (z. B. $C_6H_{13}NO_2$) und dem Massenspektrum, das die Massen der verschiedenen Fragmente des Moleküls angibt, die beim Beschuss mit einem Elektronenstrahl entstehen. Das Massenspektrum könnte z. B. einen Peak bei $m = 15$ enthalten, was der Masse eines Methyl-Fragments (CH_3) entspricht.

Die naive Version des Programms generierte alle möglichen Strukturen, die mit der Formel übereinstimmten, sagte dann voraus, welches Massenspektrum für jede Struktur beobachtet werden würde, und verglich dies mit dem tatsächlichen Spektrum. Wie zu erwarten war, ist dies selbst für Moleküle mittlerer Größe nicht effizient machbar. Die DENDRAL-Forscher zogen analytische Chemiker zu Rate und und fanden heraus, dass diese nach bekannten Peak-Mustern im Spektrum suchten, die auf gemeinsame Unterstrukturen im Molekül hindeuten. Zum Beispiel wird die folgende Regel verwendet, um eine Keton-Untergruppe (C=O) zu erkennen (deren Gewicht 28 beträgt):

> ***falls*** *M die Masse des gesamten Moleküls ist und es zwei Peaks bei x_1 und x_2 gibt, sodass (a) $x_1 + x_2 = M + 28$; (b) $x_1 - 28$ ist ein hoher Peak; (c) $x_2 - 28$ ist ein hoher Peak; und (d) mindestens eines von x_1 und x_2 ist hoch*
> ***dann*** *handelt es sich um eine Keton-Untergruppe.*

Die Erkenntnis, dass das Molekül eine bestimmte Unterstruktur enthält, reduziert die Anzahl der infrage kommenden Kandidaten enorm. DENDRAL war nach Ansicht seiner Autoren deshalb so mächtig, weil es das relevante Wissen der Massenspektroskopie nicht in Form von Grundbegriffen, sondern in Form von effizienten „Kochbuchrezepten" abbildete (Feigenbaum *et al.*, 1971). Die Bedeutung von DENDRAL lag darin, dass es das erste erfolgreiche *wissensintensive* System war: seine Expertise stammte aus einer großen Anzahl von Spezialregeln. 1971 starteten Feigenbaum und andere in Stanford das Heuristic Programming Project (HPP), um zu untersuchen, inwieweit sich die neue Methodik der **Expertensysteme** auf andere Bereiche anwenden lässt.

Die nächste große Anstrengung war das MYCIN-System zur Diagnose von Infektionskrankheiten, die mit Antibiotika behandelt werden können. Mit etwa 450 Regeln konnte MYCIN so gut wie einige Experten und deutlich besser als Assistenzärzte arbeiten. Außerdem unterschied es sich von DENDRAL in zwei wesentlichen Punkten: Erstens gab es im Gegensatz zu den DENDRAL-Regeln kein allgemeines theoretisches Modell, aus dem die MYCIN-Regeln abgeleitet werden konnten. Sie mussten aus umfangreichen Befragungen von Experten gewonnen werden. Zweitens mussten die Regeln die mit dem medizinischen Wissen verbundene Unsicherheit widerspiegeln. MYCIN enthielt ein Unsicherheitskalkül, die sogenannten **Sicherheitsfaktoren** (siehe Kapitel 13), die (damals) gut die Art und Weise abzubilden schienen, wie Ärzte die Auswirkungen von Befunden auf die Diagnose beurteilten.

Das erste kommerziell erfolgreiche Expertensystem, R1, wurde bei Digital Equipment Corporation (DEC; McDermott, 1982) in Betrieb genommen. Das Programm half bei der Konfiguration von Bestellungen für neue Computersysteme; bis 1986 sparte es dem Unternehmen geschätzte 40 Millionen US-Dollar pro Jahr. Bis 1988 hatte die KI-Gruppe von DEC 40 Expertensysteme im Einsatz, weitere waren in Vorbereitung. DuPont hatte 100 im Einsatz und 500

in der Entwicklung. Nahezu jedes größere US-Unternehmen hatte seine eigene KI-Gruppe und nutzte oder erforschte Expertensysteme.

Die Bedeutung von Fachbereichswissen zeigte sich auch im Bereich des natürlichen Sprachverständnisses (*Natural Language Understanding*, NLU). Trotz des Erfolgs von Winograds SHRDLU-System reichten seine Methoden nicht an allgemeinere Aufgaben heran: Für Probleme wie die Auflösung von Mehrdeutigkeiten verwendete es einfache Regeln, die sich auf den winzigen Anwendungsbereich der Blockwelt stützten.

Mehrere Forscher, darunter Eugene Charniak am MIT und Roger Schank in Yale, nahmen an, dass für ein robustes Sprachverständnis allgemeines Wissen über die Welt sowie eine allgemeine Methode zur Nutzung dieses Wissens nötig sei. (Schank ging noch weiter und behauptete: „So etwas wie Syntax gibt es nicht“, was viele Linguisten aufbrachte, doch es diente dazu, eine hilfreiche Diskussion anzuregen.) Schank und seine Studenten entwickelten eine Reihe von Programmen (Schank und Abelson, 1977; Wilensky, 1978; Schank und Riesbeck, 1981), die alle die Aufgabe hatten, natürliche Sprache zu verstehen. Die Betonung lag jedoch weniger auf der Sprache *per se* als vielmehr auf den Problemen der Repräsentation und des Schlussfolgerns mit dem für das Sprachverständnis erforderlichen Wissen.

Die breite Zunahme von Anwendungen auf reale Probleme führte zur Entwicklung einer breiten Palette von Repräsentations- und Schlussfolgerungswerkzeugen. Einige basierten auf Logik – zum Beispiel wurde die Sprache Prolog in Europa und Japan populär und die PLANNER-Familie in den USA. Andere, die Minskys Idee der **Frames** (1975) folgten, übernahmen einen strukturierteren Ansatz, indem sie Fakten über bestimmte Objekt- und Ereignistypen zusammenstellten und die Typen in einer großen taxonomischen Hierarchie anordneten, analog einer biologischen Taxonomie.

1981 kündigte die japanische Regierung das Projekt „Fifth Generation“ an, einen 10-Jahres-Plan für den Bau massiv paralleler, intelligenter Computer unter Prolog. Das Budget sollte 1,3 Milliarden US-Dollar übersteigen (auf heutigen Wert umgerechnet). Als Reaktion darauf gründeten die USA die Microelectronics and Computer Technology Corporation (MCC), ein Konsortium, das die nationale Wettbewerbsfähigkeit sicherstellen sollte. In beiden Fällen war die KI Teil einer breit angelegten Forschungsbemühung, unter anderem zum Chipentwurf und zu Benutzerschnittstellen. Im Vereinigten Königreich sorgte der Alvey-Bericht dafür, dass die durch den Lighthill-Bericht gestrichene Finanzierung wieder aufgenommen wurde. Keines dieser Projekte hat jedoch jemals seine ehrgeizigen Ziele in Bezug auf neue KI-Fähigkeiten oder wirtschaftliche Auswirkungen erreicht.

Insgesamt erlebte die KI-Industrie einen Anstieg von ein paar Millionen US-Dollar im Jahr 1980 auf einige Milliarden US-Dollar im Jahr 1988, außerdem gab es Hunderte von Unternehmen, die Expertensysteme, Bildverarbeitungssysteme, Roboter und darauf spezialisierte Soft- und Hardware entwickelten.

Bald darauf folgte eine Periode, die als „KI-Winter“ bezeichnet wurde, in der viele Unternehmen auf der Strecke blieben, da sie ihre extravaganten Versprechen nicht einhalten konnten. Es stellte sich heraus, dass es schwierig war, Expertensysteme für komplexe Fachgebiete zu entwickeln und zu warten – zum Teil, weil die von den Systemen verwendeten Schlussfolgerungsmethoden angesichts von Ungewissheit versagten, und zum Teil, weil die Systeme nicht aus Erfahrungen lernen konnten.

1.3.5 Die Rückkehr der neuronalen Netze (1986–heute)

Mitte der 1980er Jahre entdeckten mindestens vier verschiedene Gruppen den **Rückwärtspropagation**-Lernalgorithmus neu, der in den frühen 1960er Jahren entwickelt wurde. Der Algorithmus wurde auf viele Lernprobleme in der Informatik und Psychologie angewendet

und die weite Verbreitung der Ergebnisse im Sammelband *Parallel Distributed Processing* (Rumelhart und McClelland, 1986) sorgte für großes Aufsehen.

Diese sogenannten **konnektionistischen** Modelle wurden von einigen als direkte Konkurrenz sowohl zu den symbolischen Modellen von Newell und Simon als auch zu dem logizistischen Ansatz von McCarthy und anderen gesehen. Es scheint offensichtlich, dass Menschen auf irgendeiner Ebene Symbole manipulieren – tatsächlich legt der Anthropologe Terrence Deacon in seinem Buch *The Symbolic Species* (1997) nahe, dass dies das *definierende Merkmal* des Menschen sei. Im Gegensatz dazu hat Geoff Hinton, eine führende Figur beim Wiederaufleben neuronaler Netzwerke in den 1980er und 2010er Jahren, Symbole als den „leuchtenden Äther der KI" beschrieben – eine Anspielung auf das nicht existierende Medium, von dem viele Physiker des 19. Jahrhunderts glaubten, dass sich darin elektromagnetische Wellen ausbreiteten. Sicherlich erfüllen viele Konzepte, die wir sprachlich benennen, bei näherer Betrachtung nicht die Art von logisch definierten notwendigen und hinreichenden Bedingungen, von denen die frühen KI-Forscher hofften, sie in axiomatischer Form erfassen zu können. Es mag sein, dass konnektionistische Modelle interne Konzepte auf eine fließendere und ungenauere Art und Weise bilden, die besser an die Unordnung der realen Welt angepasst ist. Sie haben außerdem die Fähigkeit, aus Beispielen zu lernen – sie können ihren vorhergesagten Ausgabewert mit dem tatsächlichen Wert einer Problemstellung vergleichen und ihre Parameter ändern, um die Differenz zu verringern, was die Wahrscheinlichkeit erhöht, bei zukünftigen Beispielen gut abzuschneiden.

1.3.6 Probabilistisches Schlussfolgern und maschinelles Lernen (1987–heute)

Die Fragilität von Expertensystemen führte zu einem neuen wissenschaftlicheren Ansatz, der den Fokus mehr auf Wahrscheinlichkeit als auf boolesche Logik, mehr auf maschinelles Lernen als auf manuelle Programmierung und mehr auf experimentelle Ergebnisse als auf philosophische Behauptungen legte.[20] Es wurde üblicher, auf bestehenden Theorien aufzubauen anstatt völlig neue Theorien vorzuschlagen, Behauptungen auf strenge Theoreme oder solide experimentelle Methoden (Cohen, 1995) statt auf Intuition zu stützen und die Relevanz anhand realer Anwendungen statt auf Spielzeugbeispielen aufzuzeigen.

Gemeinsame Benchmark-Problemsätze wurden zur Norm, um den Fortschritt zu demonstrieren, darunter das UC Irvine Repository für Datensätze zum maschinellen Lernen, die International Planning Competition für Planungsalgorithmen, das LibriSpeech-Korpus für die Spracherkennung, der MNIST-Datensatz für die Erkennung handschriftlicher Ziffern, ImageNet und COCO für die Erkennung von Bildobjekten, SQuAD für die Beantwortung von Fragen in natürlicher Sprache, der WMT-Wettbewerb für maschinelle Übersetzung und die International SAT Competition für SAT-Solver.

Die KI wurde zum Teil als Rebellion gegen die Beschränkungen bestehender Fachgebiete wie Kontrolltheorie und Statistik gegründet, doch in dieser Periode machte sich die KI die positiven Ergebnisse dieser Gebiete zu eigen. David McAllester (1998) drückte es so aus:

> *In der Anfangszeit der KI schien es plausibel, dass neue Formen der symbolischen Berechnung, z. B. Frames und semantische Netzwerke, einen Großteil der klassischen Theorie obsolet machten. Dies führte zu einer Form von Isolationismus, in der die KI weitgehend vom Rest der Informatik getrennt wurde. Dieser Isolationismus wird derzeit aufgegeben. Man hat erkannt, dass maschinel-*

20 Von einigen wurde dieser Wandel im Streit „Neats versus Scruffies" interpretiert als Sieg der *Neats* – diejenigen, die denken, dass KI-Theorien auf mathematischer Strenge beruhen sollten – über die *Scruffies* – diejenigen, die lieber viele Ideen ausprobieren, ein paar Programme schreiben und dann beurteilen, was zu funktionieren scheint. Beide Ansätze sind wichtig. Eine Verschiebung in Richtung *Neat* bedeutet, dass das Fachgebiet ein gewisses Maß an Stabilität und Reife erreicht hat. Die derzeitige Betonung von Deep Learning könnte ein Wiederaufleben des *Scruffy*-Paradigmas darstellen.

les Lernen nicht von der Informationstheorie isoliert werden sollte, dass unsicheres Schlussfolgern nicht von stochastischer Modellierung isoliert werden sollte, dass Suche nicht von klassischer Optimierung und Steuerung isoliert werden sollte und dass automatisiertes Schlussfolgern nicht von formalen Methoden und statischer Analyse isoliert werden sollte.

Der Bereich der Spracherkennung veranschaulicht das Muster. In den 1970er Jahren wurde eine Vielzahl unterschiedlicher Architekturen und Ansätze ausprobiert. Viele davon entstanden eher ad hoc, waren fragil und funktionierten nur für wenige sorgfältig ausgewählte Beispiele. In den 1980er Jahren dominierten Ansätze mit **Hidden-Markov-Modellen** (HMMs) das Gebiet. Zwei Aspekte von HMMs sind relevant. Erstens basieren sie auf einer strengen mathematischen Theorie. Dies ermöglichte es den Sprachforschern, auf mathematischen Ergebnissen aufzubauen, die sich in anderen Bereichen über mehrere Jahrzehnte entwickelt hatten. Zweitens werden sie durch einen Trainingsprozess auf einem großen Korpus von realen Sprachdaten erzeugt. Dies stellt sicher, dass die Leistung robust ist, und in strengen Blindtests verbesserten die HMMs ihre Ergebnisse stetig. Infolgedessen schafften die Sprachtechnologie und das verwandte Gebiet der Handschrifterkennung den Übergang zu weit verbreiteten industriellen und privaten Anwendungen. Beachten Sie, dass es keinen wissenschaftlichen Anspruch gab, dass Menschen HMMs zur Spracherkennung verwenden; vielmehr boten HMMs einen mathematischen Rahmen, um das Problem zu verstehen und zu lösen. Wir werden jedoch in Abschnitt 1.3.8 sehen, dass Deep Learning dieses bequeme Narrativ ziemlich aufgemischt hat.

1988 war ein wichtiges Jahr für die Verbindung zwischen KI und anderen Gebieten, einschließlich Statistik, Operations Research, Entscheidungstheorie und Kontrolltheorie. Judea Pearls (1988) *Probabilistic Reasoning in Intelligent Systems* führte zu einer neuen Akzeptanz der Wahrscheinlichkeits- und Entscheidungstheorie in der KI. Pearls Entwicklung der **Bayes'schen Netze** brachte einen rigorosen und effizienten Formalismus zur Darstellung von unsicherem Wissen sowie praktische Algorithmen für probabilistisches Schlussfolgern hervor. Die Kapitel 12 bis 16 behandeln diesen Bereich, zusätzlich zu neueren Entwicklungen, die die Ausdruckskraft probabilistischer Formalismen stark erhöht haben; Kapitel 20 beschreibt Methoden zum Lernen von Bayes'schen Netzen und verwandten Modellen aus Daten.

Ein zweiter wichtiger Beitrag im Jahr 1988 war die Arbeit von Rich Sutton, der das Reinforcement Learning – das in Arthurs Dameprogramm in den 1950er Jahren verwendet wurde – mit der Theorie der Markov-Entscheidungsprozesse (*Markov Decision Process*, MDP) verband, die im Bereich von Operations Research entwickelt wurde. Es folgte eine Flut von Arbeiten, die die KI-Planungsforschung mit MDPs verbanden, und das Gebiet des Reinforcement Learning fand einerseits Anwendungen in der Robotik und der Prozesssteuerung und gewann andererseits tiefe theoretische Grundlagen.

Eine Folge der neu entdeckten Wertschätzung der KI für Daten, statistische Modellierung, Optimierung und maschinelles Lernen war die allmähliche Wiedervereinigung von Teilbereichen wie Computer Vision, Robotik, Spracherkennung, Multiagentensysteme und Verarbeitung natürlicher Sprache, die bis dahin etwas vom Kernbereich der KI getrennt worden waren. Der Prozess der Reintegration hat sowohl in Bezug auf Anwendungen – der Einsatz praktischer Roboter wurde zum Beispiel in dieser Zeit stark ausgeweitet – als auch in Bezug auf ein besseres theoretisches Verständnis der Kernprobleme der KI bedeutende Vorteile gebracht.

1.3.7 Big Data (2001–heute)

Bemerkenswerte Fortschritte bei der Rechenleistung sowie die Schaffung des World Wide Web haben die Erstellung sehr großer Datensätze ermöglicht – ein Phänomen, das manchmal als **Big Data** bezeichnet wird. Diese Datensätze umfassen Billionen von Wörtern, Milliarden von Bildern und Milliarden von Stunden an Sprach- und Videodaten sowie riesige Mengen

an genomischen Daten, Fahrzeugortungsdaten, Clickstream-Daten, Daten aus sozialen Netzwerken usw.

Dies hat zur Entwicklung von Lernalgorithmen geführt, die speziell für die Nutzung sehr großer Datensätze entwickelt wurden. Oft ist die überwiegende Mehrheit der Beispiele in solchen Datensätzen *unbeschriftet* (d. h. ohne *Label*); in Yarowskys (1995) einflussreichem Artikel zur Mehrdeutigkeit des Wortsinns sind beispielsweise die Vorkommen eines Worts wie „Bank" im Datensatz nicht beschriftet, sodass nicht klar ist, ob sich die Wörter auf ein Finanzunternehmen oder auf eine Sitzgelegenheit beziehen. Bei ausreichend großen Datensätzen erreichen geeignete Lernalgorithmen jedoch bei der Aufgabe zu erkennen, welche Bedeutung im jeweiligen Satz gemeint war, eine Genauigkeit von über 96 %. Darüber hinaus haben Banko und Brill (2001) argumentiert, dass die Leistungsverbesserung, die durch eine Vergrößerung des Datensatzes um zwei oder drei Größenordnungen erzielt wird, jede Verbesserung überwiegt, die durch die Optimierung des Algorithmus erreicht werden kann.

Ein ähnliches Phänomen scheint bei Aufgaben der Computer Vision wie dem Auffüllen von Leerstellen in Fotos aufzutreten – Leerstellen, die entweder durch Beschädigung oder durch das Entfernen von Ex-Freunden entstanden sind. Hays und Efros (2007) entwickelten dafür eine clevere Methode, indem sie Pixel aus ähnlichen Bildern einblendeten; sie fanden heraus, dass die Technik bei einer Datenbank mit nur Tausenden von Bildern schlecht funktionierte, aber bei Millionen von Bildern eine Qualitätsschwelle überschritt. Bald darauf löste die Verfügbarkeit von mehreren zehn Millionen Bildern in der ImageNet-Datenbank (Deng *et al.*, 2009) eine Revolution im Bereich der Computer Vision aus.

Die Verfügbarkeit von Big Data und die Hinwendung zum maschinellen Lernen verhalfen der KI zu neuer kommerzieller Attraktivität (Havenstein, 2005; Halevy *et al.*, 2009). Big Data war ein entscheidender Faktor für den Sieg von IBMs Watson-System über menschliche Gegner im Quizspiel Jeopardy! im Jahr 2011, ein Ereignis, das die öffentliche Wahrnehmung der KI stark beeinflusst hat.

1.3.8 Deep Learning (2011–heute)

Der Begriff **Deep Learning** bezieht sich auf maschinelles Lernen mit mehreren Ebenen einfacher, anpassbarer Rechenelemente. Bereits in den 1970er Jahren wurde mit solchen Netzen experimentiert, und in Form von **Convolutional Neural Networks** (CNN) hatten sie in den 1990er Jahren einige Erfolge bei der Erkennung handgeschriebener Ziffern (LeCun *et al.*, 1995). Doch erst 2011 kamen Deep-Learning-Methoden so richtig in Schwung. Dies geschah zunächst in der Spracherkennung und dann in der visuellen Objekterkennung.

Beim ImageNet-Wettbewerb 2012, bei dem Bilder in eine von tausend Kategorien (Gürteltier, Bücherregal, Korkenzieher usw.) klassifiziert werden mussten, zeigte ein Deep-Learning-System, das in der Gruppe von Geoffrey Hinton an der Universität von Toronto entwickelt wurde (Krizhevsky *et al.*, 2013), eine dramatische Verbesserung gegenüber früheren, größtenteils handindizierten Systemen. Seitdem haben Deep-Learning-Systeme die menschliche Leistung bei einigen visuellen Aufgaben übertroffen (und liegen bei einigen anderen Aufgaben zurück). Ähnliche Fortschritte wurden bei der Spracherkennung, maschinellen Übersetzung, medizinischen Diagnose und bei Spielen vermeldet. Die Verwendung eines tiefen Netzwerks zur Darstellung der Bewertungsfunktion trug zu den Siegen von AlphaGo über führende menschliche Go-Spieler bei (Silver *et al.*, 2016, 2017, 2018).

Diese bemerkenswerten Erfolge haben zu einem Wiederaufleben des Interesses an der KI bei Studenten, Unternehmen, Investoren, Regierungen, den Medien und der allgemeinen Öffentlichkeit geführt. Es scheint, als gäbe es jede Woche Neuigkeiten über eine neue KI-Anwendung, die sich der menschlichen Leistung annähert oder diese übertrifft, oft begleitet von Spekulationen über entweder beschleunigten Erfolg oder einen neuen KI-Winter.

Deep Learning ist stark auf leistungsfähige Hardware angewiesen. Während eine Standard-Computer-CPU 10^9 oder 10^{10} Operationen pro Sekunde ausführen kann, kann ein Deep-Learning-Algorithmus, der auf spezialisierter Hardware (z. B. GPU, TPU oder FPGA) läuft, zwischen 10^{14} und 10^{17} Operationen pro Sekunde verbrauchen, meist in Form von hochparallelisierten Matrix- und Vektoroperationen. Natürlich hängt Deep Learning auch von der Verfügbarkeit großer Mengen von Trainingsdaten und von einigen algorithmischen Tricks ab (siehe Kapitel 21).

1.4 State of the Art

Das Projekt „One Hundred Year Study on AI“ der Stanford-Universität (auch bekannt als AI100) beruft Expertengremien ein, um Berichte über die aktuelle Situation in der KI zu erstellen. Ihr Bericht aus dem Jahr 2016 (Stone *et al.*, 2016; Grosz und Stone, 2018) kommt zu dem Schluss, dass „ein erheblicher Anstieg der zukünftigen Nutzung von KI-Anwendungen zu erwarten ist, darunter mehr selbstfahrende Autos, Gesundheitsdiagnostik und gezielte Behandlungen sowie körperliche Unterstützung bei der Pflege älterer Menschen“ und dass „die Gesellschaft jetzt an einem entscheidenden Punkt steht, an dem es darum geht, wie KI-basierte Technologien so eingesetzt werden können, dass sie demokratische Werte wie Freiheit, Gleichheit und Transparenz fördern und nicht behindern“. AI100 erstellt auch einen **KI-Index** auf `aiindex.org`, mit dessen Hilfe der Fortschritt nachverfolgt werden kann. Einige Höhepunkte aus den Berichten 2018 und 2019 (im Vergleich zur Situation im Jahr 2000, sofern nicht anders angegeben):

- Veröffentlichungen: Die Zahl der KI-Publikationen stieg zwischen 2010 und 2019 um das 20-Fache auf etwa 20.000 pro Jahr. Die beliebteste Kategorie war maschinelles Lernen. (Die Zahl der Veröffentlichungen zum maschinellen Lernen auf `arXiv.org` hat sich von 2009 bis 2017 jedes Jahr verdoppelt.) Computer Vision und Computerlinguistik waren die nächstbeliebten Kategorien.
- Tonalität der Berichterstattung: Etwas 70 % der Nachrichtenartikel über KI sind neutral, aber Artikel mit positivem Ton sind von 12 % im Jahr 2016 auf 30 % im Jahr 2018 gestiegen. Die häufigsten Themen sind ethischer Natur: Datenschutz und algorithmische Voreingenommenheit.
- Studenten: Die Zahl der Studienanfänger ist in den USA um das 5-Fache und international um das 16-Fache gegenüber 2010 gestiegen. KI ist die beliebteste Spezialisierung innerhalb der Informatik.
- Diversität: KI-Professoren weltweit sind zu etwa 80 % männlich und zu 20 % weiblich. Ähnliche Zahlen gelten für Doktoranden und Mitarbeiter in der Industrie.
- Konferenzen: Die Teilnehmerzahl der NeurIPS stieg seit 2012 um 800 % auf 13.500 Teilnehmer. Andere Konferenzen verzeichnen ein jährliches Wachstum von etwa 30 %.
- Industrie: KI-Startups sind in den USA um das 20-Fache auf über 800 gestiegen.
- Internationalisierung: China veröffentlicht mehr Publikationen pro Jahr als die USA und etwa so viele wie ganz Europa. Hinsichtlich der Zitationshäufigkeit liegen die US-Autoren jedoch 50 % vor den chinesischen Autoren. Singapur, Brasilien, Australien, Kanada und Indien sind die Länder, die hinsichtlich der Neueinstellungen im KI-Bereich am schnellsten wachsen.
- Vision: Die Fehlerraten bei der Objekterkennung (wie sie in der LSVRC, der Large-Scale Visual Recognition Challenge, erreicht werden) verringerten sich von 28 % im Jahr 2010 auf 2 % im Jahr 2017 und übertreffen damit die menschliche Leistung. Die Genauigkeit bei der Beantwortung offener visueller Fragen (*Visual Question Answering*, VQA) hat sich seit

2015 von 55 % auf 68 % verbessert, bleibt aber hinter der menschlichen Leistung zurück, die bei 83 % liegt.

- Geschwindigkeit: Die Trainingszeit für die Bilderkennungsaufgabe ist allein in den letzten zwei Jahren um den Faktor 100 gesunken. Die Rechenleistung der Top-KI-Anwendungen verdoppelt sich alle 3,4 Monate.
- Sprache: Die Genauigkeit bei der Beantwortung von Fragen, angegeben durch das F1-Maß vom Stanford Question Answering Dataset (SQuAD), stieg von 2015 bis 2019 von 60 auf 95; bei der SQuAD 2-Variante war der Fortschritt noch schneller: von 62 auf 90 in nur einem Jahr. Beide Werte übertreffen die menschliche Leistung.
- Menschliche Benchmarks: Bis 2019 haben KI-Systeme Berichten zufolge die Leistung von Menschen in folgenden Bereichen erreicht oder übertroffen: Schach, Go, Poker, Pac-Man, Jeopardy!, ImageNet-Objekterkennung, Spracherkennung in einem begrenzten Bereich, Chinesisch-Englisch-Übersetzung in einem begrenzten Bereich, Quake III, Dota 2, StarCraft II, verschiedenen Atari-Spiele, Hautkrebserkennung, Prostatakrebserkennung, Proteinfaltung und Diagnose von diabetischer Retinopathie.

Wann (wenn überhaupt) werden KI-Systeme das Niveau erreichen, die menschlichen Leistungen bei einer breiten Fülle an Aufgaben zu übertreffen? Ford (2018) befragte KI-Experten und bekam eine große Spannbreite an Zieljahren zur Antwort – von 2029 bis 2200 –, wobei der Mittelwert das Jahr 2099 bildet. Laut einer ähnlichen Umfrage (Grace *et al.*, 2017) dachten 50 % der Befragten, dass dies bis 2066 so weit sein könnte, obwohl 10 % glaubten, dass es bereits 2025 geschehen könnte, und einige wenige antworteten „nie“. Die Experten waren auch geteilter Meinung darüber, ob wir grundlegende neue Durchbrüche oder nur Verfeinerungen aktueller Ansätze benötigen. Aber nehmen Sie ihre Vorhersagen nicht zu ernst – wie Philip Tetlock (2017) im Bereich der Vorhersage von Weltereignissen demonstriert, sind Experten darin nicht besser als Amateure.

Wie werden zukünftige KI-Systeme arbeiten? Wir können es noch nicht sagen. Wie in diesem Abschnitt beschrieben, hat das Fachgebiet im Laufe seiner Geschichte schon mehrere Perspektiven eingenommen – zuerst die kühne Idee, dass maschinelle Intelligenz überhaupt möglich ist, dann, dass sie durch die Codierung von Expertenwissen innerhalb der Logik erreicht werden kann, dann, dass probabilistische Modelle der Welt das Hauptwerkzeug sein werden, und zuletzt, dass maschinelles Lernen Modelle hervorbringen wird, die vielleicht überhaupt nicht auf einer der gut verstandenen Theorien beruhen. Die Zukunft wird zeigen, welches Modell als Nächstes kommt.

Was kann die KI heute schon leisten? Vielleicht nicht so viel, wie einige der optimistischeren Medienartikel vermuten lassen, aber doch eine ganze Menge. Hier sind einige Beispiele:

Roboterfahrzeuge: Die Geschichte der Roboterfahrzeuge reicht bis zu den funkgesteuerten Autos der 1920er Jahre zurück, doch die ersten Demonstrationen des autonomen Fahrens auf der Straße ohne spezielle Führung fanden in den 1980er Jahren statt (Kanade *et al.*, 1986; Dickmanns und Zapp, 1987). Nachdem das Fahren auf unbefestigten Straßen im Rahmen der fast 213 km langen DARPA Grand Challenge im Jahr 2005 (Thrun, 2006) und auf befestigten Straßen mit Verkehr bei der DARPA Urban Challenge 2007 erfolgreich demonstriert war, begann der Wettlauf um die Entwicklung selbstfahrender Autos ernsthaft. Im Jahr 2018 erreichten die Testfahrzeuge von Waymo die Marke von 16 Millionen gefahrenen Kilometern auf öffentlichen Straßen ohne einen schweren Unfall, wobei der menschliche Fahrer nur einmal circa alle 9.000 Kilometer die Kontrolle übernehmen musste. Bald darauf begann das Unternehmen, einen kommerziellen Robotertaxidienst anzubieten.

In der Luft sorgen autonome Starrflügler-Drohnen seit 2016 für landesweite Blutlieferungen in Ruanda. Die Quadcopter vollführen bemerkenswerte Kunstflugmanöver, erkunden Gebäude, erstellen 3-D-Karten und fügen sich zu autonomen Formationen zusammen.

Laufroboter: BigDog, ein vierbeiniger Roboter von Raibert *et al.* (2008), hat unsere Vorstellungen darüber, wie Roboter sich bewegen, auf den Kopf gestellt – kein langsamer, steifbeiniger, seitwärts gerichteter Gang der Hollywood-Filmroboter mehr, sondern eine Bewegung, die sehr der eines Tieres ähnelt und auch ein Wiederaufstehen erlaubt, wenn der Roboter geschubst wird oder auf einer Eispfütze ausrutscht. Atlas, ein humanoider Roboter, läuft nicht nur auf unebenem Terrain, sondern springt auch auf Kisten und schlägt Rückwärtssaltos (Ackerman und Guizzo, 2016).

Autonomes Planen und Scheduling: Hundert Millionen Kilometer von der Erde entfernt wurde das Remote-Agent-Programm der NASA zum ersten autonomen Planungsprogramm an Bord eines Raumfahrzeugs, das den zeitlichen Ablauf von Operationen steuerte (Jonsson *et al.*, 2000). Remote Agent generierte Pläne anhand komplexer Ziele, die vom Boden aus spezifiziert wurden, und überwachte die Ausführung dieser Pläne – Remote Agent erkannte, diagnostizierte und behob Probleme, sobald sie auftraten. Heute wird das EUROPA-Planungstoolkit (Barreiro *et al.*, 2012) für den täglichen Betrieb der Mars-Rover der NASA verwendet und das SEXTANT-System (Winternitz, 2017) ermöglicht die autonome Navigation im tiefen Weltraum, jenseits des globalen GPS-Systems.

Während der Krise am Persischen Golf im Jahr 1991 setzten die US-Streitkräfte das Programm DART (*Dynamic Analysis and Replanning Tool*; Cross und Walker, 1994) ein, um eine automatisierte logistische Planung durchzuführen und Zeitpläne für Transportaufgaben zu erstellen. Dies betraf bis zu 50.000 Fahrzeuge, Ladung und Personen gleichzeitig, und es mussten Startpunkte, Zielorte, Routen, Transportkapazitäten, Hafen- und Flugplatzkapazitäten sowie Konfliktlösungen zwischen allen Parametern berücksichtigt werden. Die DARPA (Defense Advanced Research Project Agency) stellte fest, dass sich durch diese eine Anwendung ihre 30-jährigen Investitionen in die KI mehr als ausgezahlt hatten.

Jeden Tag stellen Mitfahrunternehmen wie Uber und Kartendienste wie Google Maps Fahranweisungen für Hunderte Millionen von Nutzern bereit und berechnen schnell eine optimale Route unter Berücksichtigung der aktuellen und prognostizierten zukünftigen Verkehrsbedingungen.

Maschinelles Übersetzen: Maschinelle Online-Übersetzungssysteme ermöglichen heute das Lesen von Dokumenten in über 100 Sprachen, darunter sind die Muttersprachen von über 99 % der Menschen, und übersetzen täglich Hunderte von Milliarden von Wörtern für Hunderte von Millionen von Benutzern. Sie sind zwar nicht perfekt, aber im Allgemeinen für das Verständnis ausreichend. Bei eng verwandten Sprachen mit einer großen Menge an Trainingsdaten (wie Französisch und Englisch) liegen die Übersetzungen innerhalb eines begrenzten Fachbereichs nahe am Niveau eines Menschen (Wu *et al.*, 2016b).

Spracherkennung: Im Jahr 2017 zeigte Microsoft, dass sein System zur Konversationsspracherkennung eine Wortfehlerrate von 5,1 % erreicht hat, was der menschlichen Leistung bei der sogenannten Switchboard-Aufgabe entspricht, bei der Telefongespräche transkribiert werden (Xiong *et al.*, 2017). Etwa ein Drittel der Computerinteraktion weltweit erfolgt inzwischen per Sprache statt über die Tastatur; Skype stellt eine Echtzeit-Sprachübersetzung in zehn Sprachen zur Verfügung. Alexa, Siri, Cortana und Google bieten Assistenten an, die Fragen beantworten und Aufgaben für den Benutzer ausführen können; zum Beispiel nutzt der Dienst Google Duplex Spracherkennung und Sprachsynthese, um Restaurantreservierungen für den Benutzer vorzunehmen und eine fließende Konversation in seinem Namen durchzuführen.

Empfehlungen: Unternehmen wie Amazon, Facebook, Netflix, Spotify, YouTube, Walmart und andere nutzen maschinelles Lernen, um ihren Kunden zu empfehlen, was diesen gefallen könnte, basierend auf ihren bisherigen Erfahrungen und denen anderer Gleichgesinnter. Der Bereich der Empfehlungsdienste hat eine lange Geschichte (Resnick und Varian, 1997), verändert sich aber schnell durch neue Deep-Learning-Methoden, die sowohl Inhalte (Text, Musik, Video) als auch die Historie und Metadaten analysieren (van den Oord *et al.*, 2014; Zhang

et al., 2017). Auch die Spam-Filterung kann als eine Form der Empfehlung (bzw. Nichtempfehlung) betrachtet werden – aktuelle KI-Techniken filtern über 99,9 % des Spams heraus und E-Mail-Dienste können auch potenzielle Empfänger sowie mögliche Antworttexte empfehlen.

Spiele: Als Deep Blue 1997 den Schachweltmeister Garri Kasparow besiegte, setzten die Verfechter der menschlichen Überlegenheit ihre Hoffnungen auf Go. Piet Hut, ein Astrophysiker und Go-Enthusiast, sagte voraus, dass es „hundert Jahre dauern würde, bis ein Computer den Menschen in Go schlägt – vielleicht sogar noch länger". Doch nur 20 Jahre später übertraf AlphaGo alle menschlichen Spieler (Silver *et al.*, 2017). Der Weltmeister Ke Jie sagte: „Letztes Jahr war sein Spiel noch recht menschenähnlich. Aber dieses Jahr spielte er wie ein Gott des Go." AlphaGo profitierte vom Studium Hunderttausender vergangener Partien menschlicher Go-Spieler und vom destillierten Wissen der Go-Experten, die im Team mitarbeiteten.

Ein Nachfolgeprogramm, AlphaZero, benötigte keine Eingaben von Menschen (außer den Spielregeln) und war in der Lage, allein durch Selbstspiel zu lernen, alle Gegner, Menschen und Maschinen, bei Go, Schach und Shogi zu besiegen (Silver *et al.*, 2018). Inzwischen wurden menschliche Weltmeister von KI-Systemen bei so unterschiedlichen Spielen besiegt wie Jeopardy! (Ferrucci *et al.*, 2010), Poker (Bowling *et al.*, 2015; Moravčík *et al.*, 2017; Brown und Sandholm, 2019) und den Videospielen Dota 2 (Fernandez und Mahlmann, 2018), StarCraft II (Vinyals *et al.*, 2019) und Quake III (Jaderberg *et al.*, 2019).

Bildverstehen: Nicht zufrieden damit, die menschlichen Genauigkeit bei der anspruchsvollen ImageNet-Objekterkennungsaufgabe zu übertreffen, haben sich Forscher aus dem Bereich Computer Vision dem schwierigeren Problem der Bildbeschriftung angenommen. Beeindruckende Beispiele sind „Eine Person auf einem Motorrad auf einer unbefestigten Straße", „Zwei Pizzen auf einer Herdplatte" und „Eine Gruppe junger Leute, die spielen" (Vinyals *et al.*, 2017b). Die aktuellen Systeme sind jedoch alles andere als perfekt: Ein „Kühlschrank gefüllt mit vielen Lebensmitteln und Getränken" entpuppt sich als Parkverbotsschild, das durch viele kleine Aufkleber teilweise verdeckt wird.

Medizin: KI-Algorithmen sind inzwischen bei der Diagnose vieler Erkrankungen gleichwertig oder besser als Experten, insbesondere wenn die Diagnose auf Bildern basiert. Beispiele sind die Alzheimer-Krankheit (Ding *et al.*, 2018), metastasierender Krebs (Liu *et al.*, 2017; Esteva *et al.*, 2017), Augenkrankheiten (Gulshan *et al.*, 2016) und Hautkrankheiten (Liu *et al.*, 2019c). Eine systematische Übersichtsarbeit und Meta-Analyse (Liu *et al.*, 2019a) ergab, dass die Leistung von KI-Programmen im Durchschnitt der von medizinischem Fachpersonal gleichwertig ist. Ein aktueller Schwerpunkt in der medizinischen KI liegt in der Ermöglichung von Mensch-Maschine-Partnerschaften. So erreicht das Lyna-System eine Gesamtgenauigkeit von 99,6 % bei der Diagnose von metastasierendem Brustkrebs – besser als ein alleiniger menschlicher Experte, doch die Kombination der beiden schneidet noch besser ab (Liu *et al.*, 2018; Steiner *et al.*, 2018).

Die weit verbreitete Einführung dieser Techniken wird nun nicht durch die diagnostische Genauigkeit begrenzt, sondern durch die Notwendigkeit, eine Verbesserung der klinischen Ergebnisse nachzuweisen und Transparenz, Unvoreingenommenheit und Datenschutz zu gewährleisten (Topol, 2019). Im Jahr 2017 wurden nur zwei medizinische KI-Anwendungen von der FDA[21] zugelassen, doch bereits 2018 war diese Zahl auf 12 gestiegen und sie nimmt weiter zu.

Klimawissenschaft: Ein Team von Wissenschaftlern gewann den Gordon-Bell-Preis 2018 für ein Deep-Learning-Modell, das detaillierte Informationen über extreme Wetterereignisse entdeckt, die zuvor in Klimadaten vergraben waren. Sie nutzten einen Supercomputer mit spezialisierter GPU-Hardware, um die ExaOp-Ebene (10^{18} Operationen pro Sekunde) zu überschreiten – das erste maschinelle Lernprogramm, dem dies gelang (Kurth *et al.*, 2018). Rol-

21 Food and Drug Administration, US-amerikanische Behörde für Lebensmittel- und Arzneimittelsicherheit.

nick *et al.* (2019) präsentieren einen 60-seitigen Katalog von Möglichkeiten, wie maschinelles Lernen zur Bewältigung des Klimawandels eingesetzt werden kann.

Dies sind nur einige Beispiele für KI-Systeme, die es heute gibt: nicht Magie oder Science-Fiction – sondern Wissenschaft, Technik und Mathematik, in die dieses Buch einführt.

1.5 Risiken und Nutzen der KI

Francis Bacon, ein Philosoph, dem die Erfindung der wissenschaftlichen Methode zugeschrieben wird, bemerkte in *The Wisdom of the Ancients* (1609), dass die „mechanischen Künste von zweideutigem Nutzen sind und sowohl zum Schaden als auch zur Heilung dienen". Da die KI eine immer wichtigere Rolle in den Bereichen Wirtschaft, Soziales, Wissenschaft, Medizin, Finanzen und Militär spielt, tun wir gut daran, die Schäden und Heilmittel – modern ausgedrückt, die Risiken und Nutzen – zu bedenken, die sie mit sich bringen kann. Die hier zusammengefassten Themen werden in den Kapiteln 27 und 28 ausführlicher behandelt.

Um mit dem Nutzen zu beginnen: Einfach ausgedrückt ist unsere gesamte Zivilisation das Produkt unserer menschlichen Intelligenz. Wenn wir Zugang zu einer wesentlich höheren maschinellen Intelligenz haben, wird die Obergrenze für unsere Ambitionen erheblich angehoben. Das Potenzial von KI und Robotik, die Menschheit von niederen, sich wiederholenden Arbeiten zu befreien und die Produktion von Gütern und Dienstleistungen drastisch zu steigern, könnte eine Ära des Friedens und des Überflusses einläuten. Die Fähigkeit, die wissenschaftliche Forschung zu beschleunigen, könnte zu Heilmitteln für Krankheiten und Lösungen für den Klimawandel und die Ressourcenknappheit führen. Wie Demis Hassabis, CEO von Google DeepMind, vorgeschlagen hat: „Lösen Sie zuerst KI, dann verwenden Sie die KI, um alles andere zu lösen."

Lange bevor wir die Möglichkeit haben, „KI zu lösen", werden wir jedoch auf Risiken durch den Missbrauch von KI eingehen, ob unbeabsichtigt oder nicht. Einige davon sind bereits erkennbar, andere scheinen aufgrund der aktuellen Trends wahrscheinlich:

- *Tödliche autonome Waffen*: Diese werden von den Vereinten Nationen als Waffen definiert, die menschliche Ziele ohne menschliches Eingreifen lokalisieren, auswählen und eliminieren können. Ein Hauptsorge bei solchen Waffen ist ihre *Skalierbarkeit*: Das Fehlen der Notwendigkeit menschlicher Überwachung bedeutet, dass eine kleine Gruppe eine beliebig große Anzahl von Waffen gegen menschliche Ziele einsetzen kann, die durch jedes machbare Erkennungskriterium definiert sind. Die für autonome Waffen benötigten Technologien ähneln denen, die bei selbstfahrenden Autos zum Einsatz kommen. 2014 begannen bei den Vereinten Nationen die ersten informellen Expertendiskussionen, die dann 2017 formal in den Status der Vorbereitung eines Abkommens durch eine Gruppe von Regierungsexperten mündeten.
- *Überwachung und Einflussnahme*: Während es für Institutionen der inneren Sicherheit teuer, langwierig und manchmal rechtlich grenzwertig ist, Telefonleitungen, Videokameras, E-Mails und andere Nachrichtenkanäle zu überwachen, kann die KI (Spracherkennung, Computer Vision und natürliches Sprachverständnis) in skalierbarer Weise eingesetzt werden, um eine auf Einzelpersonen abzielende Massenüberwachung durchzuführen und bestimmte interessante Aktivitäten auszumachen. Und umgekehrt: Indem – basierend auf Techniken des maschinellen Lernens – der Informationsfluss durch die sozialen Medien auf Einzelpersonen zugeschnitten wird, kann politisches Verhalten bis zu einem gewissen Grad verändert und gesteuert werden – eine Sorge, die bei Wahlen ab dem Jahr 2016 deutlich wurde.

- *Befangene Entscheidungsfindung*: Der fahrlässige oder absichtliche Missbrauch von maschinellen Lernalgorithmen für Aufgaben wie die Bewertung von Bewährungs- und Kreditanträgen kann zu Entscheidungen führen, die aufgrund von Rasse, Geschlecht oder anderen diskriminierenden Kategorien voreingenommen sind. Oft spiegeln die Daten selbst eine weit verbreitete Voreingenommenheit in der Gesellschaft wider.
- *Auswirkungen auf die Beschäftigung*: Die Sorge, dass Maschinen Arbeitsplätze vernichten, ist Jahrhunderte alt. Die Sache ist nie einfach: Maschinen erledigen einige der Aufgaben, die sonst Menschen erledigen würden, doch sie machen Menschen auch produktiver und festigen so ihr Beschäftigungsverhältnis. Außerdem sind diese Unternehmen dann profitabler und können höhere Löhne zahlen. Einige Tätigkeiten, die sich sonst nicht lohnen würden, können dadurch wirtschaftlich rentabel werden. Ihr Einsatz führt im Allgemeinen zu steigendem Wohlstand, hat aber tendenziell den Effekt, dass sich der Wohlstand von der Arbeit zum Kapital verlagert, was die Ungleichheit weiter verschärft. Frühere technologische Fortschritte – wie die Erfindung des mechanischen Webstuhls – haben zu schwerwiegenden Einbrüchen bei der Beschäftigung geführt, aber letztendlich finden die Menschen neue Arten von Arbeit. Andererseits ist es möglich, dass die KI auch diese neuen Arten von Arbeit übernehmen wird. Dieses Thema rückt immer mehr in den Fokus von Ökonomen und Regierungen auf der ganzen Welt.
- *Sicherheitskritische Anwendungen*: Mit dem Fortschritt der KI-Techniken werden diese zunehmend in sicherheitskritischen Anwendungen mit hohem Risiko eingesetzt, z. B. beim Fahren von Autos oder bei der Verwaltung der Wasserversorgung in Städten. Es kam bereits zu tödlichen Unfällen, was die Schwierigkeit der formalen Verifizierung und statistischen Risikoanalyse für Systeme hervorhebt, die mit Techniken des maschinellen Lernens entwickelt wurden. Das Fachgebiet der KI wird technische und ethische Standards entwickeln müssen, die mindestens mit denen vergleichbar sind, die in anderen Ingenieurs- und Gesundheitsdisziplinen vorherrschen, in denen das Leben von Menschen auf dem Spiel steht.
- *Cybersicherheit*: KI-Techniken sind nützlich bei der Abwehr von Cyberangriffen, z. B. durch die Erkennung ungewöhnlicher Verhaltensmuster, aber sie werden auch dazu beitragen, dass Malware stärker, überlebens- und verbreitungsfähiger wird. So wurden beispielsweise mit Methoden des Reinforcement Learning hocheffektive Tools für automatisierte, personalisierte Erpressungs- und Phishing-Angriffe entwickelt.

Wir werden diese Themen in Abschnitt 27.3 noch einmal genauer betrachten. Wenn KI-Systeme immer leistungsfähiger werden, werden sie einige der gesellschaftlichen Rollen übernehmen, die zuvor von Menschen besetzt waren. So wie Menschen diese Rollen in der Vergangenheit genutzt haben, um Unheil zu stiften, ist zu erwarten, dass Menschen auch KI-Systeme in diesen Rollen missbrauchen werden, um noch mehr Unheil zu stiften. Alle oben genannten Beispiele betonen die Bedeutung von Lenkung und schlussendlich Regulierung. Derzeit haben die Forschungsgemeinschaft und die großen Unternehmen, die an der KI-Forschung beteiligt sind, freiwillige Selbstregulierungsprinzipien für KI-bezogene Aktivitäten entwickelt (siehe Abschnitt 27.3). Regierungen und internationale Organisationen richten Beratungsgremien ein, um für jeden spezifischen Anwendungsfall geeignete Regelungen zu erarbeiten, sich auf die wirtschaftlichen und sozialen Auswirkungen vorzubereiten und die Fähigkeiten der KI zur Lösung wichtiger gesellschaftlicher Probleme zu nutzen.

Wie sieht es auf lange Sicht aus? Werden wir das seit Langem angestrebte Ziel erreichen: die Schaffung von Intelligenz, die mit der menschlichen Intelligenz vergleichbar ist oder sie sogar übertrifft? Und wenn wir das geschafft haben, was dann?

Während eines Großteils der KI-Geschichte wurden diese Fragen vom alltäglichen Trott überschattet, der darin bestand, KI-Systeme dazu zu bewegen, irgendetwas auch nur annähernd Intelligentes zu tun. Wie bei jeder breit angelegten Disziplin hat sich die große Mehrheit

der KI-Forscher auf ein bestimmtes Teilgebiet spezialisiert, beispielsweise Spiele, Wissensrepräsentation, Vision oder Verständnis natürlicher Sprache – oft in der Annahme, dass Fortschritte im jeweiligen Teilgebiet zu den allgemeinen Zielen der KI beitragen würde. Nils Nilsson (1995), einer der leitenden Wissenschaftler des Shakey-Projekts am SRI, erinnerte das Fachgebiet an diese umfassenderen Ziele und warnte, dass die Teilgebiete Gefahr liefen, zum Selbstzweck zu werden. Später schlossen sich einige einflussreiche Begründer der KI, darunter John McCarthy (2007), Marvin Minsky (2007) und Patrick Winston (Beal und Winston, 2009), den Warnungen Nilssons an und schlugen vor, dass die KI, anstatt sich auf messbare Leistungen in spezifischen Anwendungen zu konzentrieren, zu ihren Wurzeln zurückkehren sollte, um in Herb Simons Worten „Maschinen, die denken, die lernen und die erschaffen" anzustreben. Sie nannten dieses Bestreben **Human-Level AI** oder HLAI – eine Maschine sollte in der Lage sein, alles zu lernen, was ein Mensch tun kann. Ihr erstes Symposium fand im Jahr 2004 statt (Minsky *et al.*, 2004). Ein weiteres Projekt mit ähnlichen Zielen, die **Artificial General Intelligence**-Bewegung (AGI; Goertzel und Pennachin, 2007), hielt im Jahr 2008 ihre erste Konferenz ab und organisierte das *Journal of Artificial General Intelligence*.

Etwa zur gleichen Zeit wurden Bedenken laut, dass die Erschaffung einer **künstlichen Superintelligenz** oder **ASI** – einer Intelligenz, die die menschlichen Fähigkeiten weit übertrifft – eine schlechte Idee sein könnte (Yudkowsky, 2008; Omohundro, 2008). Turing (1996) selbst äußerte sich in einer Vorlesung, die er 1951 in Manchester hielt, in Anlehnung an frühere Ideen von Samuel Butler (1863):[22]

> *Es ist wahrscheinlich, dass sobald die Methode des maschinellen Denkens begonnen hat, es nicht lange dauern wird, bis sie unsere schwachen Kräfte übertrifft. . . . An irgendeinem Punkt sollten wir also damit rechnen, dass die Maschinen die Kontrolle übernehmen, so wie es in Samuel Butlers* Erewhon *erwähnt wird.*

Diese Bedenken haben sich eben erst mit den jüngsten Fortschritten im Bereich Deep Learning, der Veröffentlichung von Büchern wie *Superintelligence* von Nick Bostrom (2014) sowie öffentlichen Äußerungen von Stephen Hawking, Bill Gates, Martin Rees und Elon Musk verstärkt.

Ein allgemeines Unbehagen bei der Idee von superintelligenten Maschinen zu verspüren, ist nur natürlich. Man könnte dies das **Gorilla-Problem** nennen: Vor etwa sieben Millionen Jahren entwickelte sich ein heute ausgestorbener Primat, von dem ein Zweig zu den Gorillas und einer zum Menschen führte. Heute sind die Gorillas nicht allzu glücklich über den menschlichen Zweig; sie haben im Grunde keine Kontrolle über ihre Zukunft. Wenn dies das Ergebnis der erfolgreichen Entwicklung von übermenschlicher KI ist – dass die Menschen die Kontrolle über ihre Zukunft abgeben –, dann sollten wir vielleicht die Arbeit an der KI einstellen und folglich auf den Nutzen verzichten, den sie bringen könnte. Dies ist die Essenz von Turings Warnung: Es ist nicht offensichtlich, dass wir Maschinen kontrollieren können, die intelligenter sind als wir.

Wäre die übermenschliche KI eine Blackbox, die aus dem Weltall kommt, dann wäre es in der Tat klug, beim Öffnen der Box Vorsicht walten zu lassen. Doch dies ist ja nicht der Fall: *Wir* entwerfen die KI-Systeme, wenn sie also am Ende „die Kontrolle übernehmen", wie Turing andeutet, wäre das das Ergebnis eines Entwurfsfehlers.

22 Noch früher, im Jahr 1847, wetterte Richard Thornton, Herausgeber des *Primitive Expounder*, gegen mechanische Rechenmaschinen: „Der Geist . . . überholt sich selbst und beseitigt die Notwendigkeit seiner eigenen Existenz, indem er Maschinen erfindet, die sein eigenes Denken übernehmen. . . . Aber wer weiß, ob solche Maschinen, wenn sie zu größerer Vollkommenheit gebracht werden, nicht einen Plan aushecken, um alle ihre eigenen Mängel zu beheben, und dann Ideen ausarbeiten, die jenseits des Wissens des sterblichen Verstands liegen!"

Um ein solches Ergebnis zu vermeiden, müssen wir die Quelle eines möglichen Fehlers verstehen. Norbert Wiener (1960), der sich angeregt sah, über die langfristige Zukunft der KI nachzudenken, nachdem er gesehen hatte, dass das Dameprogramm von Arthur Samuel lernt, seinen Schöpfer zu schlagen, sagte:

> *Wenn wir, um unsere Ziele zu erreichen, eine mechanische Apparatur benutzen, in deren Betrieb wir nicht wirksam eingreifen können . . . dann sollten wir uns ganz sicher sein, dass das Ziel, das wir der Maschine eingeben, dasselbe Ziel ist, das wir tatsächlich haben möchten.*

In vielen Kulturen gibt es Mythen von Menschen, die Götter, Geister, Magier oder Teufel um etwas bitten. Unweigerlich bekommen sie in diesen Geschichten das, worum sie wörtlich bitten, und bereuen es dann. Der dritte Wunsch, wenn es denn einen gibt, besteht in der Regel darin, die ersten beiden wieder rückgängig zu machen. Wir nennen dies das **König-Midas-Problem**: Midas, ein legendärer König in der griechischen Mythologie, bat darum, dass sich alles, was er berührte, in Gold verwandeln sollte, bereute es dann aber, nachdem er seine Speisen, Getränke und Familienmitglieder berührt hatte.[23]

Wir haben dieses Thema in Abschnitt 1.1.5 schon angeschnitten, als wir auf die Notwendigkeit einer signifikanten Modifikation des Standardmodells, feste Ziele in die Maschine einzubauen, hingewiesen haben. Die Lösung für Wieners Dilemma besteht darin, überhaupt kein festes „Ziel in die Maschine einzugeben“. Stattdessen wollen wir Maschinen, die danach streben, menschliche Ziele zu erreichen, aber wissen, dass sie nicht mit Sicherheit wissen, was genau diese Ziele sind.

Es ist vielleicht ein unglücklicher Zustand, dass fast die gesamte bisherige KI-Forschung innerhalb des Standardmodells durchgeführt wurde, was bedeutet, dass fast das gesamte technische Material in diesem Buch diesen intellektuellen Rahmen widerspiegelt. Es gibt jedoch einige frühe Ergebnisse innerhalb des neuen Rahmens. In Kapitel 16 zeigen wir, dass eine Maschine genau dann einen positiven Anreiz hat, sich abschalten zu lassen, wenn sie sich über das menschliche Ziel unsicher ist. In Kapitel 18 formulieren und untersuchen wir **Assistenzspiele**, die mathematisch die Situation beschreiben, in der ein Mensch ein Ziel hat und eine Maschine versucht, dieses Ziel zu erreichen, aber zunächst unsicher ist, wie das Ziel lautet. In Kapitel 22 erläutern wir die Methoden des **Inverse Reinforcement Learning (IRL)**, die es Maschinen ermöglichen, mehr über menschliche Präferenzen zu lernen, indem sie beobachten, welche Entscheidungen die Menschen treffen. In Kapitel 27 gehen wir auf zwei der Hauptschwierigkeiten ein: die erste ist, dass unsere Entscheidungen von unseren Präferenzen, d. h. von einer sehr komplexen kognitiven Architektur abhängen, die nur schwer umkehrbar ist; die zweite, dass wir Menschen möglicherweise nicht von vornherein konsistente Präferenzen haben – weder individuell noch als Gruppe –, sodass es möglicherweise nicht klar ist, was KI-Systeme für uns tun *sollten*.

23 Midas hätte besser daran getan, wenn er sich an die grundlegenden Prinzipien der Sicherheit gehalten hätte und eine „Rückgängig“- und eine „Pause“-Taste in seinen Wunsch eingebaut hätte.

ZUSAMMENFASSUNG

In diesem Kapitel wird die KI definiert und der kulturelle Hintergrund dargelegt, vor dem sie sich entwickelt hat. Die wichtigsten Punkte sind:

- Unterschiedliche Menschen gehen mit unterschiedlichen Zielen an die KI heran. Zwei wichtige Fragen, die gestellt werden sollten, sind: Geht es Ihnen um das Denken oder um das Verhalten? Wollen Sie den Menschen modellieren, oder versuchen Sie, optimale Ergebnisse zu erzielen?
- Legt man das sogenannte Standardmodell zugrunde, dann geht es in der KI hauptsächlich um **rationales Handeln**. Ein idealer **intelligenter Agent** ergreift die bestmögliche Handlung in einer Situation. Wir untersuchen das Problem, Agenten zu bauen, die in diesem Sinne intelligent sind.
- Diese einfache Idee muss in zwei Punkten verfeinert werden: Erstens ist die Fähigkeit eines jeden Agenten, ob menschlich oder nicht, rationale Handlungen zu wählen, durch die effiziente Machbarkeit der Berechnung begrenzt; zweitens muss das Konzept einer Maschine, die ein bestimmtes Ziel verfolgt, durch das einer Maschine ersetzt werden, die Ziele verfolgt, die den Menschen zugute kommen, wobei die Maschine aber nicht sicher weiß, wie diese Ziele genau lauten.
- Philosophen (deren diesbezügliche Arbeiten bis 400 v. Chr. zurückgehen) schufen den Nährboden für die KI, indem sie anregten, den Verstand in gewisser Weise wie eine Maschine zu betrachten, der mit Wissen arbeitet, das in einer internen Sprache codiert ist, und Gedanken dazu verwendet werden können, um zu entscheiden, welche Handlungen ausgeführt werden sollen.
- Mathematiker lieferten die Werkzeuge, um sowohl Aussagen mit logischer Gewissheit als auch unsichere, probabilistische Aussagen zu handhaben. Sie legten auch den Grundstein für das Verständnis von Berechenbarkeit und Schlussfolgern über Algorithmen.
- Wirtschaftswissenschaftler formalisierten das Problem, Entscheidungen zu treffen, die den erwarteten Nutzen für den Entscheidungsträger maximieren.
- Neurowissenschaftler entdeckten einige Fakten darüber, wie das Gehirn funktioniert und auf welche Weise es Computern ähnlich ist bzw. sich von ihnen unterscheidet.
- Psychologen übernahmen die Idee, dass Menschen und Tiere als informationsverarbeitende Maschinen betrachtet werden können. Linguisten zeigten, dass der Sprachgebrauch in dieses Modell passt.
- Computerspezialisten lieferten auf der Hardwareseite immer leistungsfähigere Maschinen, die KI-Anwendungen möglich machen, und machten sie auf der Softwareseite besser nutzbar.
- Die Kontrolltheorie befasst sich mit dem Entwurf von Geräten, die auf der Grundlage von Feedback aus der Umgebung optimal agieren. Anfangs unterschieden sich die mathematischen Werkzeuge der Kontrolltheorie völlig von denen der KI, doch die Fachgebiete nähern sich nun einander an.
- In der Geschichte der KI gab es Zyklen von Erfolg, unangebrachtem Optimismus und daraus resultierenden Rückschritten sowohl in der Begeisterung als auch in der Finanzierung. Doch es gab auch Zyklen, in denen neue, kreative Ansätze eingeführt und die besten davon systematisch verfeinert wurden.

- Die KI hat sich im Vergleich zu den ersten Jahrzehnten erheblich weiterentwickelt, sowohl theoretisch als auch methodisch. Da die Probleme, mit denen sich die KI befasst, immer komplexer wurden, hat sich das Fachgebiet von der booleschen Logik zur probabilistischen Schlussfolgerung und vom selbst erarbeiteten Wissen zum maschinellen Lernen aus Daten entwickelt. Dies hat zu Verbesserungen der Fähigkeiten realer Systeme und zu einer stärkeren Integration mit anderen Disziplinen geführt.
- Da KI-Systeme in der realen Welt Anwendung finden, ist es notwendig geworden, eine große Bandbreite von Risiken und ethischen Konsequenzen zu berücksichtigen.
- Längerfristig stehen wir vor dem schwierigen Problem, superintelligente KI-Systeme zu kontrollieren, die sich möglicherweise auf unvorhersehbare Weise weiterentwickeln. Die Lösung dieses Problems scheint eine Änderung unserer Vorstellung von KI zu erfordern.

Bibliografische und historische Anmerkungen

Eine umfassende Geschichte der KI wird von Nils Nilsson (2009) gegeben, einem der frühen Pioniere des Fachgebiets. Pedro Domingos (2015) und Melanie Mitchell (2019) geben Übersichten über maschinelles Lernen für ein allgemeines Publikum und Kai-Fu Lee (2018) beschreibt das Rennen um die internationale Führung in der KI. Martin Ford (2018) interviewt 23 führende KI-Forscher.

Die wichtigsten KI-Fachgesellschaften sind die *Association for the Advancement of Artificial Intelligence* (AAAI), die *ACM Special Interest Group in Artificial Intelligence* (SIGAI, früher SIGART), die *European Association for AI* und die *Society for Artificial Intelligence and Simulation of Behaviour* (AISB). Die *Partnership on AI* bringt viele kommerzielle und gemeinnützige Organisationen zusammen, die sich mit den ethischen und sozialen Auswirkungen der KI beschäftigen. Das *AI Magazine* der AAAI enthält viele aktuelle Artikel und Tutorials und auf ihrer Website `aaai.org` finden sich Nachrichten, Tutorials und Hintergrundinformationen.

Die neuesten Arbeiten erscheinen in den Tagungsbändern der großen KI-Konferenzen: der *International Joint Conference on AI* (IJCAI), der jährlichen *European Conference on AI* (ECAI) und der AAAI-Konferenz. Maschinelles Lernen wird durch die *International Conference on Machine Learning* (ICML, jetzt ICMLA) und die Tagung *Neural Information Processing Systems* (NeurIPS) abgedeckt. Die wichtigsten Fachzeitschriften für allgemeine KI sind *Artificial Intelligence*, *Computational Intelligence*, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *IEEE Intelligent Systems* und das *Journal of Artificial Intelligence Research*. Es gibt auch viele Konferenzen und Zeitschriften, die sich mit speziellen Bereichen befassen, diese geben wir in den entsprechenden Kapiteln an.

Intelligente Agenten

2

In diesem Kapitel diskutieren wir das Wesen von Agenten, ob perfekt oder nicht, die Vielfalt von Umgebungen sowie die daraus resultierende Menagerie von Agententypen.

In Kapitel 1 haben wir das Konzept der **rationalen Agenten** als zentral für unseren Ansatz zur künstlichen Intelligenz identifiziert. In diesem Kapitel wollen wir diesen Begriff nun konkretisieren. Wir werden sehen, dass das Konzept der Rationalität auf eine Vielzahl von Agenten angewendet werden kann, die in jeder erdenklichen Umgebung agieren. Wir möchten in diesem Buch anhand dieses Konzepts einige Entwurfsprinzipien für den Aufbau erfolgreicher Agenten entwickeln – Systeme, die mit Fug und Recht als **intelligent** bezeichnet werden können.

Wir beginnen mit der Untersuchung von Agenten, Umgebungen und der Verknüpfung zwischen ihnen. Die Beobachtung, dass sich einige Agenten besser verhalten als andere, führt zwangsläufig zu der Idee eines rationalen Agenten – eines Agenten, der sich so gut wie möglich verhält. Wie gut sich ein Agent verhalten kann, hängt von der Art der Umgebung ab; manche Umgebungen sind schwieriger als andere. Wir werden Umgebungen grob in Kategorien einordnen und zeigen, wie die Eigenschaften einer Umgebung den Entwurf von Agenten beeinflussen, die für diese Umgebung geeignet sind. Wir beschreiben eine Reihe von grundlegenden „Gerüsten“ für den Entwurf von Agenten, die wir im weiteren Verlauf des Buchs weiter ausbauen.

2.1 Agenten und Umgebungen

Als **Agent** kann im Prinzip alles angesehen werden, was seine **Umgebung** mithilfe von **Sensoren** wahrnimmt und durch **Aktuatoren** auf diese Umgebung einwirkt. Dieses einfache Konzept ist in ► Abbildung 2.1 dargestellt. Ein menschlicher Agent hat Augen, Ohren und andere Organe als Sensoren sowie Hände, Beine, Vokaltrakt usw. als Aktuatoren. Ein Roboter-Agent könnte Kameras und Infrarotentfernungsmesser als Sensoren und verschiedene Motoren als Aktuatoren besitzen. Ein Software-Agent empfängt Dateiinhalte, Netzwerkpakete und menschliche Eingaben (Tastatur/Maus/Touchscreen/Stimme) als sensorische Eingaben und wirkt auf die Umgebung ein, indem er Dateien schreibt, Netzwerkpakete

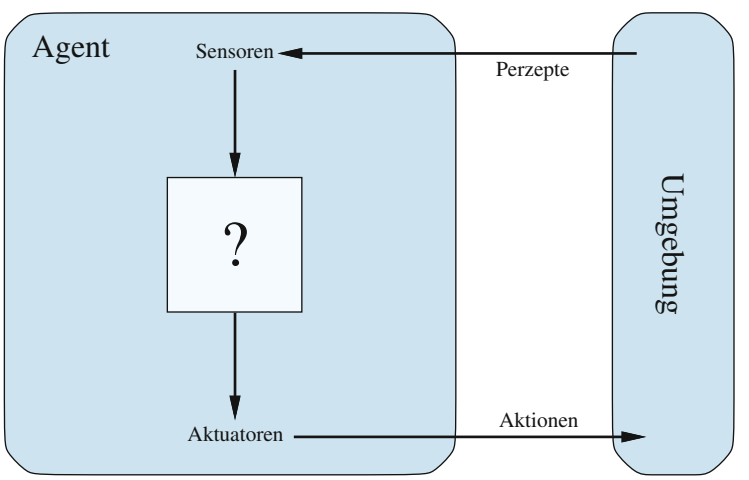


Abbildung 2.1: Agenten interagieren mit der Umgebung durch Sensoren und Aktuatoren.

sendet, Informationen auf dem Bildschirm anzeigt oder Töne erzeugt. Die Umgebung könnte alles sein – das gesamte Universum! In der Praxis ist eine Umgebung allerdings nur der Teil des Universums, dessen Zustand für den Entwurf dieses Agenten relevant ist – also der Teil, der beeinflusst, was der Agent wahrnimmt, und der umgekehrt von den Aktionen des Agenten beeinflusst wird.

Wir verwenden den Begriff **Perzept**, um den Inhalt zu bezeichnen, den die Sensoren eines Agenten wahrnehmen. Die **Perzeptfolge** eines Agenten ist die vollständige Historie von allem, was der Agent jemals wahrgenommen hat. Im Allgemeinen *kann die Auswahl einer Aktion durch den Agenten zu jedem beliebigen Zeitpunkt von seinem integrierten Wissen und von der gesamten bisherigen Perzeptfolge abhängen, aber nicht von etwas, das der Agent nicht wahrgenommen hat.* Indem wir die Auswahl der Aktion eines Agenten für jede mögliche Perzeptfolge spezifizieren, haben wir mehr oder weniger alles gesagt, was es über den Agenten zu sagen gibt. Mathematisch betrachtet sagen wir, dass das Verhalten eines Agenten durch die **Agentenfunktion** beschrieben wird, die jede beliebige Perzeptfolge auf eine Aktion abbildet.

Wir können uns vorstellen, die Agentenfunktion, die einen beliebigen Agenten beschreibt, *tabellarisch* wiederzugeben. Für die meisten Agenten wäre dies allerdings eine sehr große Tabelle – eigentlich unendlich groß, es sei denn, wir begrenzen die Länge der Perzeptfolgen, die wir berücksichtigen wollen. Wir können im Prinzip für einen Agenten, mit dem wir experimentieren wollen, diese Tabelle dadurch erstellen, indem wir alle möglichen Perzeptfolgen ausprobieren und dann aufzeichnen, welche Aktionen der Agent als Reaktion darauf ausführt.[1] Die Tabelle ist natürlich eine *externe* Charakterisierung des Agenten. *Intern* wird die Agentenfunktion für einen künstlichen Agenten durch ein **Agentenprogramm** implementiert. Es ist wichtig, diese beiden Konzepte auseinanderzuhalten: Die Agentenfunktion ist eine abstrakte mathematische Beschreibung – das Agentenprogramm ist eine konkrete Implementierung, die in einem physischen System ausgeführt wird.

Zur Veranschaulichung dieser Ideen verwenden wir ein einfaches Beispiel – die Staubsaugerwelt, die aus einem staubsaugenden Roboter-Agenten besteht, der sich in einer Welt aus Feldern befindet, die jeweils entweder schmutzig oder sauber sein können. ► Abbildung 2.2 zeigt eine Konfiguration mit nur zwei Feldern, *A* und *B*. Der Staubsauger-Agent nimmt wahr, in welchem Feld er sich befindet und ob es Schmutz in diesem Feld gibt. Der Agent startet im Feld *A*. Die möglichen Aktionen sind: nach rechts gehen, nach links gehen, den Schmutz auf-

1 Wählt der Agent seine Aktionen nach dem Zufallsprinzip aus, dann müssten wir jede Folge mehrfach ausprobieren, um die Wahrscheinlichkeit jeder Aktion zu ermitteln. Man könnte meinen, zufällig Aktionen auszuwählen sei ziemlich dumm, doch wir werden später in diesem Kapitel sehen, dass es durchaus sehr intelligent sein kann.

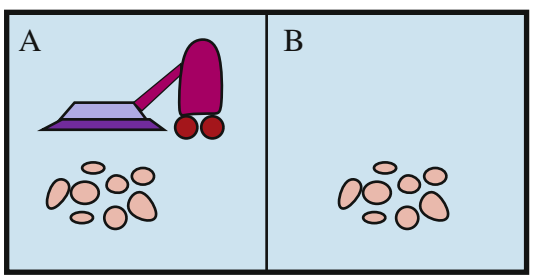


Abbildung 2.2: Eine Staubsaugerwelt mit nur zwei Feldern. Jedes Feld kann sauber oder schmutzig sein und der Agent kann sich nach links oder rechts bewegen und kann das Feld säubern, in dem er sich befindet. In verschiedenen Versionen der Staubsaugerwelt kann man unterschiedliche Regeln darüber implementieren, was der Agent wahrnehmen kann, ob seine Aktionen immer erfolgreich sind usw.

Perzeptfolge	Aktion
[*A*, *Sauber*]	*Rechts*
[*A*, *Schmutzig*]	*Saugen*
[*B*, *Sauber*]	*Links*
[*B*, *Schmutzig*]	*Saugen*
[*A*, *Sauber*], [*A*, *Sauber*]	*Rechts*
[*A*, *Sauber*], [*A*, *Schmutzig*]	*Saugen*
⋮	⋮
[*A*, *Sauber*], [*A*, *Sauber*], [*A*, *Sauber*]	*Rechts*
[*A*, *Sauber*], [*A*, *Sauber*], [*A*, *Schmutzig*]	*Saugen*
⋮	⋮

Abbildung 2.3: Ausschnitt der tabellarischen Darstellung einer einfachen Agentenfunktion für die in Abbildung 2.2 gezeigte Staubsaugerwelt. Der Agent säubert das aktuelle Feld, wenn es schmutzig ist, ansonsten geht er zum anderen Feld. Beachten Sie, dass die Größe der Tabelle unbegrenzt ist, solange es keine Beschränkung der Länge möglicher Perzeptfolgen gibt.

saugen oder nichts tun.[2] Eine sehr einfache Agentenfunktion ist die folgende: Wenn das aktuelle Feld schmutzig ist, dann sauge dort; andernfalls bewege dich auf das andere Feld. Einen Teil der tabellarischen Darstellung dieser Agentenfunktion sehen Sie in ► Abbildung 2.3 und ein Agentenprogramm, das die Funktion implementiert, ist in ► Abbildung 2.8 auf S. 75 zu sehen.

Man kann anhand der Tabelle in ► Abbildung 2.3 sehen, dass verschiedene Agenten der Staubsaugerwelt einfach dadurch definiert werden können, dass man die rechte Spalte unterschiedlich ausfüllt. Die offensichtliche Frage ist also: *Wie kann die Tabelle richtig ausgefüllt werden?* Mit anderen Worten: Was macht einen Agenten gut oder schlecht, intelligent oder dumm? Im nächsten Abschnitt werden wir diese Fragen beantworten.

Bevor wir diesen Abschnitt abschließen, sollten wir betonen, dass das Konzept des Agenten als Werkzeug zur Analyse von Systemen gedacht ist und nicht als absolute Charakterisierung, die die Welt in Agenten und Nichtagenten einteilt. Man könnte sogar einen Taschenrechner als Agenten betrachten, der wählt, bei der Perzeptfolge „2 + 2 = “ die Aktion „4“ anzuzeigen, doch eine solche Analyse würde uns kaum dabei helfen, den Taschenrechner zu verstehen. In gewissem Sinne lassen sich sämtliche Bereiche der Technik als Entwerfen von Artefakten betrachten, die mit der Welt interagieren; die KI arbeitet (nach Ansicht der Autoren) am interessantesten Ende des Spektrums, an dem die Artefakte über beträchtliche Rechenressourcen verfügen und die Aufgabenumgebung nichttriviale Entscheidungsfindung voraussetzt.

2.2 Gutes Verhalten: das Konzept der Rationalität

Ein **rationaler Agent** ist ein Agent, der das Richtige tut. Offensichtlich ist es besser, das Richtige zu tun, als das Falsche zu tun, aber was bedeutet es, „das Richtige“ zu tun?

2 Bei einem echten Roboter gäbe es Aktionen wie „nach rechts gehen“ und „nach links gehen“ wahrscheinlich nicht. Stattdessen würden die Aktionen lauten: „Räder vorwärts drehen“ und „Räder rückwärts drehen“. Wir haben uns jedoch für diese Benennung entschieden, um die Aktionen leichter auf der Seite nachvollziehen zu können, nicht für die einfachere Implementierung eines echten Roboters.

2.2.1 Leistungsmessung

Innerhalb der Ethik wurden verschiedene Vorstellungen vom „Richtigen“ entwickelt, aber die KI hat sich im Allgemeinen an ein Konzept gehalten, das man als **Konsequenzialismus** bezeichnet: Wir bewerten das Verhalten eines Agenten anhand seiner Konsequenzen. Wird ein Agent in eine Umgebung hineingestellt, dann generiert er anhand der empfangenen Perzepte eine Folge von Handlungen. Diese Handlungssequenz führt dazu, dass die Umgebung eine Abfolge von Zuständen durchläuft. Ist die Folge wünschenswert, so hat der Agent eine gute Leistung erbracht. Dieser Begriff der Erwünschtheit wird durch ein **Leistungsmaß** erfasst, das jede beliebige Folge von Umgebungszuständen bewertet.

Menschen haben eigene Wünsche und Präferenzen, daher geht es bei dem Konzept der Rationalität, angewandt auf Menschen, darum, wie erfolgreich Handlungen gewählt werden, aus denen dann Sequenzen von Umgebungszuständen entstehen, die *aus ihrer Sicht* wünschenswert sind. Maschinen hingegen haben *keine* eigenen Wünsche und Präferenzen; die Leistungsbewertung findet, zumindest anfangs, im Kopf des Entwicklers der Maschine oder im Kopf der Benutzer statt, für die die Maschine konzipiert ist. Wir werden sehen, dass einige Agentenentwürfe eine explizite Repräsentation (einer Version) des Leistungsmaßes besitzen, während in anderen Entwürfen das Leistungsmaß vollständig implizit ist – der Agent mag vielleicht „das Richtige“ tun, doch er weiß nicht, warum.

Erinnern wir uns an Norbert Wieners Warnung, die uns mahnt sicherzustellen, dass „das Ziel, das wir der Maschine eingeben, dasselbe Ziel ist, das wir tatsächlich haben möchten“ (S. 57), so stellen wir fest, dass es ziemlich schwierig sein kann, ein Leistungsmaß korrekt zu formulieren. Betrachten wir zum Beispiel den Staubsauger-Agenten aus dem vorigen Abschnitt. Wir könnten empfehlen, die Leistung anhand der Menge an Schmutz zu messen, die in einer Acht-Stunden-Schicht aufgesaugt wird. Bei einem rationalen Agenten ist das, was Sie verlangen, natürlich auch das, was Sie bekommen. Ein rationaler Agent kann dieses Leistungsmaß maximieren, indem er den Schmutz aufsaugt, dann wieder alles auf den Boden kippt, dann wieder saugt und so weiter. Ein geeigneteres Leistungsmaß würde den Agenten dafür belohnen, einen sauberen Boden zu haben. Zum Beispiel könnte für jedes saubere Feld pro Zeitschritt ein Punkt vergeben werden (vielleicht mit einem Abzug für verbrauchte Elektrizität und erzeugten Lärm). *Generell ist es besser, die Leistungsmaße danach zu gestalten, was man in der Umgebung tatsächlich erreichen möchte, und nicht danach, wie man meint, dass sich der Agent verhalten soll.*

Selbst wenn die offensichtlichen Fallstricke umgangen werden, bleiben einige knifflige Probleme bestehen. Zum Beispiel basiert der Begriff „sauberer Boden“ im vorigen Absatz auf der durchschnittlichen Sauberkeit im Laufe der Zeit. Doch dieselbe durchschnittliche Sauberkeit kann von zwei unterschiedlichen Agenten erreicht werden, von denen einer die ganze Zeit über eine mittelmäßige Arbeit leistet, während der andere energisch saugt, aber lange Pausen macht. Was vorzuziehen ist, erscheint zwar wie eine Spitzfindigkeit aus der Hausmeisterwissenschaft, ist aber tatsächlich eine tiefgreifende philosophische Frage mit weitreichenden Implikationen. Was ist besser – ein unbekümmertes Leben mit Höhen und Tiefen oder eine sichere, aber eintönige Existenz? Was ist besser – eine Wirtschaft, in der alle in mäßiger Armut leben, oder eine, in der einige im Überfluss leben, während andere sehr arm sind? Wir überlassen dem eifrigen Leser diese Fragen als Übung.

Wir werden in diesem Buch weitestgehend voraussetzen, dass das Leistungsmaß korrekt angegeben werden kann. Aus den oben genannten Gründen müssen wir jedoch die Möglichkeit in Kauf nehmen, dass wir der Maschine ein falschen Ziel angeben – das ist genau das König-Midas-Problem, das wir auf S. 57 beschrieben haben. Außerdem können wir bei der Entwicklung einer Software, deren Kopien an verschiedene Benutzer verteilt werden, nicht die genauen Präferenzen jedes einzelnen Benutzers vorhersehen. Aus diesem Grund müssen wir möglicherweise Agenten erstellen, bei denen das richtige Leistungsmaß anfangs noch unsicher ist und im Laufe der Zeit verfeinert wird; solche Agenten werden in den Kapiteln 16, 18 und 22 beschrieben.

2.2.2 Rationalität

Was zu einem bestimmten Zeitpunkt rational ist, hängt von vier Dingen ab:

- dem Leistungsmaß, das das Erfolgskriterium definiert,
- dem Vorwissen des Agenten über die Umgebung,
- den Aktionen, die der Agent ausführen kann,
- der bisherigen Perzeptfolge des Agenten.

Diese Punkte führen uns zu einer **Definition eines rationalen Agenten**:

> Ein rationaler Agent soll für jede mögliche Perzeptfolge eine Aktion wählen, von der erwartet wird, dass sie sein Leistungsmaß maximiert, wenn die Ergebnisse der Perzeptfolge sowie jegliches Vorwissen des Agenten berücksichtigt werden.

Betrachten wir den einfachen Staubsauger-Agenten, der ein Feld säubert, wenn es schmutzig ist, und sich andernfalls zum nächsten Feld bewegt – dies ist die in ► Abbildung 2.3 tabellarisch dargestellte Agentenfunktion. Handelt es sich dabei um einen rationalen Agenten? Das kommt darauf an! Zuerst müssen wir angeben, was das Leistungsmaß ist, was über die Umgebung bekannt ist und welche Sensoren und Aktuatoren der Agent hat. Wir wollen Folgendes annehmen:

- Das Leistungsmaß vergibt einen Punkt für jedes saubere Feld pro Zeitschritt, und zwar über eine „Lebensdauer" von 1.000 Zeitschritten.
- Die „Geografie" der Umgebung ist *a priori* bekannt (► Abbildung 2.2), aber die Schmutzverteilung und die Anfangsposition des Agenten sind es nicht. Saubere Felder bleiben sauber und durch Saugen wird das aktuelle Feld gereinigt. Die Aktionen *Rechts* und *Links* bewegen den Agenten um ein Feld in der entsprechenden Richtung weiter, es sei denn, dies würde den Agenten aus der Umgebung herausführen; in diesem Fall bleibt der Agent, wo er ist.
- Die einzigen verfügbaren Aktionen sind *Rechts*, *Links* und *Saugen*.
- Der Agent nimmt seine Position korrekt wahr und erkennt, ob sein Standort schmutzig ist.

Unter diesen Umständen ist der Agent tatsächlich rational; seine erwartete Leistung ist mindestens so gut wie die jedes anderen Agenten.

Man kann leicht nachvollziehen, dass derselbe Agent unter anderen Umständen irrational wäre. Zum Beispiel wird der Agent, sobald der gesamte Schmutz beseitigt ist, unnötig hin- und herpendeln – beinhaltet das Leistungsmaß eine Strafe von einem Punkt für jede Bewegung, so wird der Agent schlecht abschneiden. Ein für diesen Fall besserer Agent würde nichts tun, sobald er sicher ist, dass alle Felder sauber sind. Falls saubere Felder wieder schmutzig werden können, sollte der Agent sie gelegentlich überprüfen und bei Bedarf erneut reinigen. Ist die Geografie der Umgebung unbekannt, dann muss der Agent sie **erkunden**. In Übung 2.2 auf der Website sollen Sie Agenten für solche Fälle entwerfen.

2.2.3 Allwissenheit, Lernen und Autonomie

Wir müssen sorgfältig zwischen Rationalität und **Allwissenheit** unterscheiden. Ein allwissender Agent kennt das *tatsächliche* Ergebnis seiner Aktionen und kann entsprechend handeln – in der Realität ist Allwissenheit allerdings unmöglich. Betrachten Sie das folgende Beispiel: Ich gehe eines Tages auf den Champs Elysées spazieren und sehe einen alten Freund auf der anderen Straßenseite. Es gibt um mich herum keinen Straßenverkehr und ich bin nicht anderweitig beschäftigt, also ist es rational von mir, die Straße zu überqueren. Währenddessen

fällt in 10.000 Meter Höhe eine Frachttür aus einem über mir fliegenden Passagierflugzeug[3] und bevor ich es auf die andere Straßenseite schaffe, bin ich platt. War es irrational von mir, die Straße zu überqueren? Es ist unwahrscheinlich, dass in meiner Todesanzeige steht: „Der Dummkopf starb beim Versuch, die Straße zu überqueren."

Dieses Beispiel zeigt, dass Rationalität nicht mit Perfektion gleichzusetzen ist. Rationalität maximiert die *erwartete* Leistung, während Perfektion die *tatsächliche* Leistung maximiert. Wenn wir von der Forderung nach Perfektion abrücken, ist das nicht nur eine Frage der Fairness gegenüber Agenten. Der springende Punkt ist: Wenn wir von einem Agenten erwarten, dass er das tut, was sich im Nachhinein als die beste Aktion herausstellt, dann ist es unmöglich, einen Agenten zu entwerfen, der diese Vorgabe erfüllt – es sei denn, wir verbessern die Leistung von Kristallkugeln oder Zeitmaschinen.

Für unsere Definition von Rationalität benötigen wir also keine Allwissenheit, denn die rationale Wahl hängt nur von der *bisherigen* Perzeptfolge ab. Wir müssen außerdem sicherstellen, dass wir dem Agenten nicht versehentlich erlaubt haben, ausgesprochen dumme Aktivitäten auszuführen. Schaut ein Agent z. B. nicht nach rechts und nach links, bevor er eine viel befahrene Straße überquert, dann wird er seiner Perzeptfolge nicht entnehmen können, dass sich ein großer Lkw mit hoher Geschwindigkeit nähert. Besagt unsere Definition von Rationalität, dass es jetzt in Ordnung ist, die Straße zu überqueren? Weit gefehlt!

Erstens wäre es angesichts dieser nicht informativen Perzeptfolge nicht rational, die Straße zu überqueren: Das Unfallrisiko beim Überqueren, ohne nach rechts und links zu schauen, ist zu groß. Zweitens sollte ein rationaler Agent die Aktion „Schauen" wählen, bevor er auf die Straße tritt, weil dies hilft, die erwartete Leistung zu maximieren. Das Ausführen von Aktionen, *um zukünftige Perzepte zu verändern* – manchmal auch als **Informationsbeschaffung** bezeichnet –, ist ein wichtiger Teil der Rationalität und wird in Kapitel 16 ausführlich behandelt. Ein zweites Beispiel für das Sammeln von Informationen ist die **Exploration**, die ein Staubsauger-Agent in einer zunächst unbekannten Umgebung durchführen muss.

Unsere Definition verlangt von einem rationalen Agenten nicht nur, dass er Informationen sammelt, sondern er soll auch so viel wie möglich **lernen** aus dem, was er wahrnimmt. Die anfängliche Konfiguration des Agenten könnte ein gewisses Vorwissen über die Umgebung widerspiegeln, doch sobald der Agent an Erfahrung gewinnt, kann dieses Wissen modifiziert und erweitert werden. Es gibt Extremfälle, in denen die Umgebung *a priori* vollständig bekannt und vollständig vorhersagbar ist. In solchen Fällen muss der Agent nichts wahrnehmen oder lernen; er handelt einfach korrekt, wenn er entsprechend programmiert wird.

Agenten, die weder wahrnehmen noch lernen können, sind natürlich anfällig. Nehmen wir den einfachen Mistkäfer: Nachdem er sein Nest gegraben und seine Eier abgelegt hat, holt er eine Mistkugel von einem nahe gelegenen Haufen, um den Eingang zu verstopfen. Wird ihm die Mistkugel *auf dem Weg* dorthin abgenommen, dann setzt der Käfer seine Aufgabe trotzdem fort, er verstopft das Nest pantomimisch mit der nicht vorhandenen Mistkugel und bemerkt nicht einmal, dass sie fehlt. Die Evolution hat eine Annahme in das Verhalten des Käfers eingebaut und wenn diese verletzt wird, resultiert daraus ein erfolgloses Verhalten.

Ein wenig intelligenter ist die Sphex, eine Gattung der Grabwespe. Das Sphex-Weibchen gräbt eine Höhle, kommt heraus, sticht eine Raupe und schleppt sie vor die Höhle, betritt die Höhle erneut, um zu prüfen, ob alles in Ordnung ist, zieht die Raupe hinein und legt ihre Eier ab. Die Raupe dient als Nahrungsquelle für die Larven, die aus den Eiern schlüpfen. So weit, so gut. Doch wenn ein Entomologe die Raupe um ein paar Zentimeter verschiebt, während die Sphex ihren Kontrollgang durchführt, dann kehrt sie zum Schritt „die Raupe vor die Höhle schleppen" ihres Plans zurück und wird diesen unverändert fortsetzen, kontrolliert die Höhle erneut, anstatt die Raupe direkt in die Höhle zu ziehen – und das auch, nachdem die Raupe

3 Siehe N. Henderson, „New door latches urged for Boeing 747 jumbo jets", *Washington Post*, 24. August 1989.

mehrfach verschoben wurde. Die Sphex ist nicht in der Lage zu lernen, dass ihr angeborener Plan fehlschlägt, und wird ihn daher nicht ändern.

Verlässt sich ein Agent auf das Vorwissen seines Entwicklers anstatt auf seine eigenen Wahrnehmungen und Lernprozesse, dann sprechen wir davon, dass es dem Agenten an **Autonomie** mangelt. Ein rationaler Agent sollte autonom sein – er sollte so viel wie möglich lernen, um unvollständiges oder falsches Vorwissen zu kompensieren. Ein Staubsauger-Agent, der lernt vorherzusehen, wo und wann zusätzlicher Schmutz auftaucht, wird beispielsweise besser abschneiden als ein Agent, der das nicht kann.

In der Praxis ist es selten erforderlich, dass ein Agent von Anfang an völlig autonom ist: Wenn der Agent wenig oder gar keine Erfahrung hat, müsste er zufällig handeln, falls ihm sein Entwickler keine Hilfestellung mitgegeben hat. Genauso wie die Evolution Tiere mit ausreichend angeborenen Reflexen ausstattet, damit sie lange genug überleben, bis sie selbst lernen, wäre es vernünftig, einen künstlichen intelligenten Agenten mit einem gewissen Anfangswissen sowie einer Fähigkeit zum Lernen auszustatten. Nach ausreichender Erfahrung mit seiner Umgebung kann das Verhalten eines rationalen Agenten praktisch *unabhängig* von seinem Vorwissen werden. Die Einbeziehung des Lernens ermöglicht es also, einen einzigen rationalen Agenten zu entwerfen, der in einer Vielzahl von Umgebungen erfolgreich ist.

2.3 Arten von Umgebungen

Da wir nun eine Definition der Rationalität haben, sind wir fast so weit, über die Entwicklung rationaler Agenten nachzudenken. Zunächst müssen wir uns jedoch Gedanken über **Aufgabenumgebungen** machen, die im Wesentlichen die „Probleme“ darstellen, deren „Lösungen“ die rationalen Agenten sind. Als Erstes zeigen wir, wie man eine Aufgabenumgebung spezifiziert, und illustrieren diesen Prozess anhand einer Reihe von Beispielen. Dann stellen wir Aufgabenumgebungen in unterschiedlichen Varianten vor. Die Art der Aufgabenumgebung wirkt sich direkt darauf aus, welcher Entwurf für das Agentenprogramm geeignet ist.

2.3.1 Spezifizieren der Aufgabenumgebung

In unserer Diskussion zur Rationalität des einfachen Staubsauger-Agenten mussten wir das Leistungsmaß, die Umgebung sowie die Aktuatoren und Sensoren des Agenten angeben. Dies alles fassen wir nun unter dem Oberbegriff der **Aufgabenumgebung** zusammen. Da wir gerne Akronyme benutzen, nennen wir dies die **PEAS-Beschreibung** (von **P**erformance, **E**nvironment, **A**ctuators, **S**ensors – Leistung, Umgebung, Aktuatoren, Sensoren). Beim Entwurf eines Agenten muss es immer der erste Schritt sein, die Aufgabenumgebung so vollständig wie möglich zu spezifizieren.

Die Staubsaugerwelt war ein einfaches Beispiel – betrachten wir nun ein komplexeres Problem: einen automatisierten Taxifahrer. ► Abbildung 2.4 fasst die PEAS-Beschreibung für die Aufgabenumgebung des Taxis zusammen. In den folgenden Abschnitten besprechen wir jedes Element im Detail.

Als Erstes fragen wir, welches **Leistungsmaß** wir für unseren automatisierten Fahrer anstreben möchten. Zu den wünschenswerten Eigenschaften gehören: das korrekte Ziel erreichen, den Kraftstoffverbrauch und den Verschleiß minimieren, die Fahrzeit oder -kosten minimieren, Verstöße gegen Verkehrsregeln sowie die Behinderungen anderer Fahrer minimieren, Sicherheit und Fahrgastkomfort sowie den Gewinn maximieren. Offensichtlich stehen einige dieser Ziele im Widerspruch zueinander, sodass Kompromisse eingegangen werden müssen.

Die nächste Frage ist: Wie sieht die **Fahrumgebung** des Taxis aus? Jeder Taxifahrer muss mit einer Vielzahl von Straßen zurechtkommen, von ländlichen Wegen und städtischen Gassen

Agententyp	Leistungsmaß	Umgebung	Aktuatoren	Sensoren
Taxifahrer	Sicher, schnell, legale Fahrweise, komfortable Fahrt, maximiert Gewinne, minimiert Auswirkung auf andere Verkehrsteilnehmer	Straßen, anderer Verkehr, Polizei, Fußgänger, Fahrgäste, Wetter	Lenkrad, Gaspedal, Bremspedal, Blinker, Hupe, Bildschirmanzeige, Sprache	Kameras, Radar, Tachometer, GPS, Motorsensoren, Beschleunigungsmesser, Mikrofone, Touchscreenbildschirm

Abbildung 2.4: PEAS-Beschreibung der Aufgabenumgebung für einen automatisierten Taxifahrer.

bis zu vielspurigen Autobahnen. Auf den Straßen gibt es andere Verkehrsteilnehmer, Fußgänger, kreuzende Wildtiere, Baustellen, Polizeiautos, Pfützen und Schlaglöcher. Außerdem muss das Taxi mit potenziellen und tatsächlichen Fahrgästen kommunizieren. Und es gibt einige Wahlmöglichkeiten. Das Taxi könnte in Südkalifornien eingesetzt werden, wo Schnee selten ein Thema ist – oder in Alaska, wo es selten keins ist. Es könnte immer auf der rechten Seite fahren – oder möchten wir, dass es flexibel genug ist, um auf der linken Seite zu fahren, wenn es sich in Großbritannien oder Japan befindet? Je eingeschränkter die Umgebung ist, desto einfacher ist natürlich das Entwurfsproblem.

Die **Aktuatoren** des automatisierten Taxis sind im Prinzip dieselben, die auch einem menschlichen Fahrer zur Verfügung stehen: Bedienung des Motors durch das Gaspedal und Steuerung über Lenkung und Bremsen. Darüber hinaus wird eine Ausgabe auf einem Bildschirm oder über einen Sprachsynthesizer benötigt, damit das Taxi mit den Fahrgästen sprechen kann, und vielleicht eine Möglichkeit, um mit anderen Fahrzeugen – höflich oder anderweitig – zu kommunizieren.

Zu den grundlegenden **Sensoren** für das Taxi werden eine oder mehrere Videokameras gehören, damit es etwas sehen kann, sowie Lidar- und Ultraschallsensoren, um Abstände zu anderen Autos und Hindernissen zu erkennen. Um Geschwindigkeitsüberschreitungen zu vermeiden, sollte das Taxi einen Tachometer haben, und um das Fahrzeug vor allem in Kurven richtig zu steuern, sollte es einen Beschleunigungsmesser besitzen. Zur Feststellung des mechanischen Zustands des Fahrzeugs, benötigt es die üblichen Sensoren für Motor, Kraftstoff und Elektrik. Wie viele menschliche Fahrer wird es möglicherweise auf GPS-Signale zugreifen wollen, damit es sich nicht verirrt. Schließlich benötigt es Touchscreen- oder Spracheingaben, damit der Fahrgast sein Ziel mitteilen kann.

In ► Abbildung 2.5 skizzieren wir die grundlegenden PEAS-Elemente für eine Reihe zusätzlicher Agententypen. Weitere Beispiele finden Sie in Übung 2.5 auf der Website. Die Beispiele umfassen sowohl physische als auch virtuelle Umgebungen. Beachten Sie, dass virtuelle Aufgabenumgebungen genauso komplex sein können wie die „reale“ Welt: Ein **Software-Agent** (auch als Software-Roboter oder **Softbot** bezeichnet), der auf Auktions- und Wiederverkaufsplattformen handelt, hat beispielsweise mit Millionen anderer Benutzer und Milliarden von Objekten zu tun, darunter viele mit realen Bildern.

2.3.2 Eigenschaften von Aufgabenumgebungen

Die Bandbreite der Aufgabenumgebungen, die in der KI auftreten können, ist offensichtlich riesig. Wir können jedoch eine relativ kleine Anzahl an Dimensionen identifizieren, entlang derer Aufgabenumgebungen kategorisiert werden können. Diese Dimensionen legen zu einem großen Teil den geeigneten Entwurf für einen Agenten sowie die Anwendbarkeit der grundlegenden Familien von Verfahren für die Implementierung fest. Zunächst listen wir die Dimensionen auf, dann analysieren wir mehrere Aufgabenumgebungen, um die Konzepte zu verdeutlichen. Die Definitionen hier sind informell; in späteren Kapiteln finden Sie genauere Aussagen und Beispiele für die einzelnen Umgebungen.

Agententyp	Leistungsmaß	Umgebung	Aktuatoren	Sensoren
Medizinisches Diagnosesystem	Gesunder Patient, verringerte Kosten	Patient, Krankenhaus, Mitarbeiter	Anzeige von Fragen, Untersuchungen, Diagnosen, Behandlungen	Touchscreen/ Spracheingabe von Symptomen und Befunden
Satellitenbild-Analysesystem	Korrekte Objekt- und Geländekategorisierung	Satellit in der Umlaufbahn, Downlink, Wetter	Anzeige der Szenenkategorisierung	Hochauflösende Digitalkamera
Pack-Roboter für Werkstücke	Prozentsatz der Werkstücke in korrekten Behältern	Förderband mit Werkstücken; Behälter	Gelenkarm und -hand	Kamera, taktile und Gelenkwinkelsensoren
Raffinerie-Controller	Reinheit, Ausbeute, Sicherheit	Raffinerie, Rohmaterialien, Arbeiter	Ventile, Pumpen, Heizungen, Rührwerke, Anzeigen	Temperatur-, Druck-, Durchfluss- und chemische Sensoren
Interaktiver Englischlehrer	Punktzahl/Note des Schülers im Test	Schülergruppe, Prüfungskommission	Anzeige von Übungen, Feedback, Sprache	Tastatureingabe, Stimme

Abbildung 2.5: Beispiele für Agententypen und ihre PEAS-Beschreibungen.

Vollständig beobachtbar – teilweise beobachtbar: Wenn die Sensoren eines Agenten ihm zu jedem Zeitpunkt Zugriff auf den vollständigen Zustand der Umgebung gestatten, dann sagen wir, dass die Aufgabenumgebung vollständig beobachtbar ist. Eine Aufgabenumgebung ist faktisch vollständig beobachtbar, wenn die Sensoren alle Aspekte erfassen, die für die Wahl der Aktion *relevant* sind; die Relevanz hängt wiederum vom Leistungsmaß ab. Vollständig beobachtbare Umgebungen sind praktisch, weil der Agent keinen internen Zustand verwalten muss, um die Welt im Auge zu behalten. Eine Umgebung ist möglicherweise nur teilweise beobachtbar, weil die Sensoren verrauscht und ungenau sind oder weil Teile des Zustands einfach nicht in den Sensordaten enthalten sind – zum Beispiel kann ein Staubsauger-Agent, der mit einem lokalen Schmutzsensor ausgestattet ist, nicht erkennen, ob sich in anderen Feldern Schmutz befindet, und ein automatisiertes Taxi kann nicht sehen, was andere Fahrer denken oder planen. Wenn der Agent überhaupt keine Sensoren hat, dann ist die Umgebung **unbeobachtbar**. Man könnte meinen, dass die Lage des Agenten in solchen Fällen hoffnungslos ist, doch wie wir in Kapitel 4 besprechen werden, können die Ziele des Agenten dennoch – manchmal sogar absolut – erreichbar sein.

Einzelagent – Multiagent: Die Unterscheidung zwischen Einzelagenten- und Multiagentenumgebungen scheint einfach zu sein. Zum Beispiel befindet sich ein Agent, der alleine ein Kreuzworträtsel löst, offensichtlich in einer Einzelagentenumgebung, während ein Agent, der Schach spielt, in einer Zwei-Agenten-Umgebung ist. Es gibt jedoch einige subtile Probleme. Wir haben zwar beschrieben, wie eine Entität als Agent betrachtet werden *kann*, aber wir haben nicht geklärt, welche Entitäten als Agenten betrachtet werden *müssen*. Muss ein Agent A (z. B. der Taxifahrer) ein Objekt B (ein anderes Fahrzeug) als Agent behandeln oder kann er es einfach als ein Objekt ansehen, das sich gemäß den physikalischen Gesetzen verhält, wie Wellen am Strand oder Blätter, die im Wind wehen? Das wesentliche Unterscheidungsmerkmal ist, ob das Verhalten von B am besten als Maximierung eines Leistungsmaßes beschrieben wird, dessen Wert vom Verhalten von Agent A abhängt.

Beim Schachspiel zum Beispiel versucht die gegnerische Entität B ihr Leistungsmaß zu maximieren, was nach den Regeln des Schachspiels das Leistungsmaß von Agent A minimiert. Schach ist also eine **konkurrierende** Multiagentenumgebung. In der Taxi-Umgebung dagegen verbessert die Vermeidung von Kollisionen das Leistungsmaß aller Agenten, es handelt sich

also um eine partiell **kooperative** Multiagentenumgebung. Sie besitzt aber auch konkurrierende Aspekte, z. B. weil jeweils nur ein Auto einen Parkplatz belegen kann.

Die Probleme beim Entwurf von Agenten in Multiagentenumgebungen sind häufig etwas andere als in Einzelagentenumgebungen; zum Beispiel erweist sich Kommunikation in Multiagentenumgebungen oft als rationales Verhalten; in einigen konkurrierenden Umgebungen ist zufälliges Verhalten rational, weil es die Fallstricke der Vorhersagbarkeit vermeidet.

Deterministisch – nichtdeterministisch. Wenn der nächste Zustand der Umgebung vollständig durch den aktuellen Zustand sowie durch die von dem oder den Agenten ausgeführte Aktion festgelegt wird, dann sagen wir, dass die Umgebung deterministisch ist; andernfalls ist sie nichtdeterministisch. Im Prinzip muss sich ein Agent in einer vollständig beobachtbaren, deterministischen Umgebung keine Gedanken über Unsicherheit machen. Ist die Umgebung jedoch nur teilweise beobachtbar ist, könnte sie als nichtdeterministisch *erscheinen*.

Die meisten realen Situationen sind so komplex, dass es unmöglich ist, den Überblick über alle unbeobachteten Aspekte zu behalten; für praktische Zwecke müssen sie als nichtdeterministisch angesehen werden. Die Taxi-Umgebung ist in diesem Sinne eindeutig nichtdeterministisch, weil man das Verkehrsgeschehen nie genau vorhersagen kann; außerdem kann es passieren, dass die Reifen unerwartet platzen oder der Motor ohne Vorwarnung ausfällt. Die Staubsaugerwelt, so wie wir sie beschrieben haben, ist deterministisch, aber Varianten können nichtdeterministische Elemente enthalten, z. B. Schmutz, der zufällig dazukommt, oder ein unzuverlässiger Saugmechanismus (Übung 2.16 auf der Website).

Eine letzte Anmerkung: Der Begriff **stochastisch** wird von manchen als Synonym für „nichtdeterministisch“ verwendet, doch wir unterscheiden zwischen den beiden Begriffen – wir sagen, dass ein Modell der Umwelt stochastisch ist, wenn es explizit mit Wahrscheinlichkeiten arbeitet (z. B. „es besteht eine 25%ige Chance, dass es morgen regnet“), und „nichtdeterministisch“, wenn die Möglichkeiten aufgelistet werden, ohne dass sie quantifiziert werden (z. B. „es besteht die Chance, dass es morgen regnet“).

Episodisch – sequenziell: In einer episodischen Aufgabenumgebung ist die Erfahrung des Agenten in atomare Episoden unterteilt. In jeder Episode empfängt der Agent ein Perzept und führt dann eine einzelne Aktion aus. Entscheidend ist, dass die nächste Episode nicht von den Aktionen der vorhergehenden Episoden abhängt. Viele Klassifikationsaufgaben sind episodisch. Beispielsweise basiert jede Entscheidung eines Agenten, der defekte Teile auf einem Fließband erkennen soll, nur auf dem aktuellen Teil und ist unabhängig von früheren Entscheidungen; außerdem hat die aktuelle Entscheidung keinen Einfluss darauf, ob das nächste Teil defekt ist. In sequenziellen Umgebungen hingegen könnte die aktuelle Entscheidung alle zukünftigen Entscheidungen beeinflussen.[4] Schach und Taxifahren sind sequenziell: In beiden Fällen können kurzfristige Aktionen langfristige Konsequenzen haben. Episodische Umgebungen sind viel einfacher als sequenzielle Umgebungen, weil der Agent nicht vorausdenken muss.

Statisch – dynamisch: Wenn sich die Umgebung ändern kann, während ein Agent eine Entscheidung trifft, dann sprechen wir davon, dass die Umgebung für diesen Agenten dynamisch ist; andernfalls ist sie statisch. Statische Umgebungen sind einfach zu handhaben, da der Agent die Welt nicht ständig beobachten muss, während er sich für eine Aktion entscheidet, und er muss sich auch keine Gedanken über den Zeitablauf machen. Dynamische Umgebungen hingegen fragen den Agenten ständig, was er tun möchte; hat er sich noch nicht entschieden, so wird das als Entscheidung gewertet, nichts zu tun. Ändert sich im Laufe der Zeit nicht die Umgebung, sondern das Leistungsmaß des Agenten, dann sagen wir, dass die Umgebung **semidynamisch** ist. Taxifahren ist offensichtlich dynamisch: Die anderen Autos

4 Das Wort „sequenziell“ wird in der Informatik auch als Antonym von „parallel“ verwendet. Die beiden Bedeutungen sind aber weitgehend unabhängig voneinander.

und das Taxi selbst bewegen sich, während der Fahralgorithmus berechnet, was als Nächstes zu tun ist. Schach ist, wenn es mit einer Uhr gespielt wird, semidynamisch. Kreuzworträtsellösen ist statisch.

Diskret – stetig: Die Unterscheidung diskret/stetig bezieht sich auf den *Zustand* der Umgebung, auf die Art und Weise, wie die *Zeit* gehandhabt wird, sowie auf die *Perzepte* und *Aktionen* des Agenten. Zum Beispiel hat die Schachumgebung eine endliche Anzahl unterschiedlicher Zustände (von der Uhr abgesehen). Außerdem besitzt Schach eine diskrete Menge von Perzepten und Aktionen. Taxifahren ist eine Problemstellung mit stetigen Zuständen und stetiger Zeit: Die Geschwindigkeit und der Standort des Taxis und der anderen Fahrzeuge durchlaufen einen Bereich stetiger Werte, und zwar gleichmäßig über der Zeit. Auch die Aktionen des Taxifahrens sind stetig (Lenkwinkel usw.). Die Eingaben von Digitalkameras sind streng genommen diskret, werden aber typischerweise so behandelt, als würden sie stetig variierende Intensitäten und Positionen darstellen.

Bekannt – unbekannt: Streng genommen bezieht sich diese Unterscheidung nicht auf die Umgebung selbst, sondern auf den Wissensstand des Agenten (oder seines Entwicklers) über die „physikalischen Gesetze" der Umgebung. In einer bekannten Umgebung sind die Ergebnisse (oder Ergebniswahrscheinlichkeiten, wenn die Umgebung nichtdeterministisch ist) für alle Aktionen gegeben. Ist die Umgebung unbekannt, dann muss der Agent natürlich lernen, wie sie funktioniert, um gute Entscheidungen treffen zu können.

Die Unterscheidung zwischen bekannten und unbekannten Umgebungen ist nicht dieselbe wie die zwischen vollständig und teilweise beobachtbaren Umgebungen. Es ist durchaus möglich, dass eine *bekannte* Umgebung nur *teilweise* beobachtbar ist – bei einem Solitär-Kartenspiel zum Beispiel kenne ich zwar die Regeln, kann aber nicht die Karten sehen, die noch nicht umgedreht wurden. Umgekehrt kann eine *unbekannte* Umgebung *vollständig* beobachtbar sein – bei einem neuen Videospiel zeigt der Bildschirm vielleicht den gesamten Spielzustand an, aber ich weiß erst, was die Tasten bedeuten, wenn ich sie ausprobiere.

Wie auf S. 65 erwähnt, kann das Leistungsmaß selbst unbekannt sein, entweder weil der Entwickler nicht sicher ist, wie er es korrekt angeben soll, oder weil der Endanwender – auf dessen Präferenzen es ankommt – noch nicht bekannt ist. Ein Taxifahrer weiß zum Beispiel normalerweise nicht, ob ein neuer Fahrgast eine gemächliche oder schnelle Fahrt, einen vorsichtigen oder aggressiven Fahrstil bevorzugt. Ein virtueller persönlicher Assistent weiß zu Beginn nichts über die persönlichen Vorlieben seines neuen Besitzers. In solchen Fällen kann der Agent durch weitere Interaktionen mit dem Entwickler oder Benutzer mehr über das Leistungsmaß erfahren. Dies wiederum legt nahe, dass die Aufgabenumgebung notwendigerweise als Multiagentenumgebung betrachtet wird.

Der schwierigste Fall ist eine *teilweise beobachtbare*, *nichtdeterministische*, *sequenzielle*, *dynamische*, *stetige* und *unbekannte Multiagenten*umgebung. Taxifahren ist in all diesen Aspekten schwierig, außer dass die Umgebung des Fahrers größtenteils bekannt ist. Das Fahren eines Mietwagens in einem neuen Land mit nicht vertrauter Geografie, anderen Verkehrsregeln und nervösen Fahrgästen ist viel aufregender.

In ► Abbildung 2.6 sind die Eigenschaften einiger vertrauter Umgebungen aufgeführt. Beachten Sie, dass die Eigenschaften nicht immer ganz eindeutig sind. Zum Beispiel haben wir die medizinische Diagnose als Aufgabe für einen einzelnen Agenten aufgeführt, weil der Krankheitsprozess bei einem Patienten nicht nutzbringend als Agent modelliert werden kann; aber ein medizinisches Diagnosesystem muss eventuell auch mit widerspenstigen Patienten und skeptischem Personal umgehen, sodass die Umgebung durchaus auch Aspekte einer Multiagentenumgebung haben könnte. Außerdem ist die medizinische Diagnose episodisch, wenn man die Aufgabe so versteht, dass eine Diagnose anhand einer Liste von Symptomen ausgewählt wird; das Problem ist aber sequenziell, wenn die Aufgabe das Vorschlagen einer Reihe

Aufgabenumgebung	Beobachtbar	Agenten	Deterministisch	Episodisch	Statisch	Diskret
Kreuzworträtsel	Vollständig	Einzel	Deterministisch	Sequenziell	Statisch	Diskret
Schach mit Uhr	Vollständig	Multi	Deterministisch	Sequenziell	Semidyn.	Diskret
Poker	Teilweise	Multi	Nichtdeterministisch	Sequenziell	Statisch	Diskret
Backgammon	Vollständig	Multi	Nichtdeterministisch	Sequenziell	Statisch	Diskret
Taxifahren	Teilweise	Multi	Nichtdeterministisch	Sequenziell	Dynamisch	Stetig
Medizinische Diagnose	Teilweise	Einzel	Nichtdeterministisch	Sequenziell	Dynamisch	Stetig
Bildanalyse	Vollständig	Einzel	Deterministisch	Episodisch	Semidyn.	Stetig
Pack-Roboter für Werkstücke	Teilweise	Einzel	Nichtdeterministisch	Episodisch	Dynamisch	Stetig
Raffinerie-Controller	Teilweise	Einzel	Nichtdeterministisch	Sequenziell	Dynamisch	Stetig
Interaktiver Englischlehrer	Teilweise	Multi	Nichtdeterministisch	Sequenziell	Dynamisch	Diskret

Abbildung 2.6: Beispiele für Aufgabenumgebungen und ihre Eigenschaften.

von Tests, die Bewertung des Fortschritts im Verlauf der Behandlung, die Behandlung mehrerer Patienten usw. beinhalten kann.

Wir haben keine Spalte „bekannt/unbekannt“ aufgenommen, weil dies, wie bereits erwähnt, streng genommen keine Eigenschaft der Umgebung ist. Bei einigen Umgebungen, wie z. B. Schach und Poker, ist es recht einfach, den Agenten mit vollständigem Regelwissen auszustatten, doch es ist trotzdem interessant zu überlegen, wie ein Agent lernen könnte, die Spiele auch ohne dieses Wissen zu meistern.

Der zu diesem Buch bereitgestellt Code (`aima.cs.berkeley.edu`) enthält Implementierungen für mehrere Umgebungen sowie einen allgemeinen Umgebungssimulator zur Bewertung der Leistung eines Agenten. Experimente werden oft nicht für eine einzelne Umgebung durchgeführt, sondern für viele Umgebungen, die aus einer **Umgebungsklasse** stammen. Um beispielsweise einen Taxifahrer im simulierten Verkehr zu bewerten, möchten wir gerne viele Simulationen mit unterschiedlichen Verkehrs-, Licht- und Wetterbedingungen durchführen. Wir sind dann an der durchschnittlichen Leistung des Agenten in der Umgebungsklasse interessiert.

2.4 Die Struktur von Agenten

Bisher haben wir uns bei unseren Ausführungen über Agenten auf die Beschreibung ihres *Verhaltens* konzentriert – die Aktion, die nach einer bestimmten Folge von Perzepten ausgeführt wird. Jetzt müssen wir in den sauren Apfel beißen und darüber sprechen, wie das Innenleben eines Agenten aussieht. Die Aufgabe der KI ist es, ein **Agentenprogramm** zu entwerfen, das die Agentenfunktion implementiert – die Abbildung von Perzepten auf Aktionen. Wir gehen davon aus, dass dieses Programm auf irgendeiner Art Rechengerät mit physischen Sensoren und Aktuatoren läuft – wir nennen dies die **Agentenarchitektur**:

$$Agent = Architektur + Programm$$

Natürlich muss das Programm, das wir auswählen, für die Architektur geeignet sein. Wenn das Programm Aktionen wie *Gehen* empfehlen soll, dann sollte die Architektur auch irgendeine Form von Beinen haben. Bei der Architektur könnte es sich um einen gewöhnlichen PC handeln oder um ein Roboterauto mit mehreren integrierten Computern, Kameras und anderen Sensoren. Im Allgemeinen stellt die Architektur dem Programm die Perzepte der

Sensoren zur Verfügung, führt das Programm aus und gibt die vom Programm vorgeschlagenen Aktionen an die Aktuatoren weiter, sobald sie generiert wurden. Der größte Teil dieses Buchs befasst sich mit dem Entwurf von Agentenprogrammen, während sich die Kapitel 25 und 26 direkt mit den Sensoren und Aktuatoren befassen.

2.4.1 Agentenprogramme

Die Agentenprogramme, die wir in diesem Buch entwerfen, haben alle das gleiche Grundgerüst: Sie nehmen das aktuelle Perzept als Eingabe von den Sensoren entgegen und geben eine Aktion an die Aktuatoren zurück.[5] Beachten Sie den Unterschied zwischen dem Agentenprogramm, das das aktuelle Perzept als Eingabe entgegennimmt, und der Agentenfunktion, die von der gesamten Perzepthistorie abhängen kann. Dem Agentenprogramm bleibt nichts anderes übrig, als lediglich das aktuelle Perzept als Eingabe zu übernehmen, weil sonst nichts von der Umgebung zur Verfügung gestellt wird – sollen die Aktionen des Agenten von der gesamten Perzeptfolge abhängen, dann muss sich der Agent die Perzepte merken.

Wir beschreiben die Agentenprogramme in der einfachen Pseudocode-Sprache, die in Anhang B definiert ist. (Der online bereitgestellte Code enthält Implementierungen in echten Programmiersprachen.) ► Abbildung 2.7 zeigt zum Beispiel ein eher triviales Agentenprogramm, das die Perzeptfolge verfolgt und sie dann verwendet, um eine passende Aktion anhand des entsprechenden Indexes aus der Tabelle herauszusuchen und zu entscheiden, was zu tun ist. Die Tabelle – ein Beispiel dafür ist in ► Abbildung 2.3 für die Staubsaugerwelt dargestellt – repräsentiert explizit die Agentenfunktion, die wiederum das Agentenprogramm verkörpert. Um auf diese Weise einen rationalen Agenten zu erstellen, müssen wir als Entwickler eine Tabelle aufbauen, die für jede mögliche Perzeptfolge die entsprechende Aktion enthält.

```
function TABLE-DRIVEN-AGENT(percept) returns eine Aktion
  persistent: percepts, eine Folge, anfangs leer
              table, eine Tabelle mit Aktionen, durch Perzeptfolgen indiziert,
              anfangs vollständig spezifiziert

  percept am Ende von percepts anhängen
  action ← LOOKUP(percepts, table)
  return action
```

Abbildung 2.7: Das Programm TABLE-DRIVEN-AGENT wird für jedes neue Perzept aufgerufen und gibt jeweils eine Aktion zurück. Es verwaltet die vollständige Perzeptfolge im Speicher.

Es ist sehr aufschlussreich, sich zu überlegen, warum der tabellengesteuerte Ansatz zur Entwicklung von Agenten zum Scheitern verurteilt ist. Sei $\mathcal{P}$ die Menge der möglichen Perzepte und T die Lebensdauer des Agenten (die Gesamtzahl der Perzepte, die er entgegennimmt). Die Nachschlagetabelle (LOOKUP) wird dann $\sum_{t=1}^{T} |\mathcal{P}|^t$ Einträge enthalten. Betrachten wir das automatisierte Taxi: Die visuelle Eingabe einer einzelnen Kamera (acht Kameras sind üblich) trifft mit einer Rate von etwa 70 Megabyte pro Sekunde ein (30 Bilder pro Sekunde, 1.080×720 Pixel mit 24 Bit Farbinformationen). Damit kommen wir auf eine Nachschlagetabelle mit über $10^{600.000.000.000}$ Einträgen für eine Stunde Fahrzeit. Sogar die Nachschlagetabelle für Schach – ein winziges, braves Fragment der realen Welt – hat (wie sich herausstellt) mindestens 10^{150} Einträge. Im Vergleich dazu beträgt die Anzahl der Atome im beobacht-

5 Es gibt noch andere Möglichkeiten für das Grundgerüst eines Agentenprogramms; zum Beispiel könnten wir die Agentenprogramme als **Coroutinen** ausführen, die asynchron zu der Umgebung laufen. Jede solche Coroutine hat einen Eingabe- und einen Ausgabeport und besteht aus einer Schleife, die Perzepte vom Eingabeport liest und Aktionen auf den Ausgabeport schreibt.

baren Universum weniger als 10^{80}. Die erschreckende Größe dieser Tabellen bedeutet, dass (a) kein physischer Agent in diesem Universum genügend Platz hat, die Tabelle zu speichern; (b) kein Entwickler die Zeit hätte, die Tabelle zu erstellen; und (c) kein Agent jemals alle richtigen Tabelleneinträge aus seiner Erfahrung lernen könnte.

Trotz alledem erledigt TABLE-DRIVEN-AGENT *genau das*, was wir wollen, vorausgesetzt, die Tabelle ist korrekt ausgefüllt: Es implementiert die gewünschte Agentenfunktion. *Die zentrale Herausforderung der KI besteht darin herauszufinden, wie man Programme schreibt, die rationales Verhalten so weit wie möglich aus einem recht kleinen Programm statt aus einer riesigen Tabelle erzeugen.*

Es gibt viele Beispiele, die zeigen, dass dies auch in anderen Bereichen erfolgreich möglich ist: So wurden die riesigen Quadratwurzeltabellen, die vor den 1970er Jahren von Ingenieuren und Schulkindern verwendet wurden, durch ein fünfzeiliges Programm ersetzt, welches das Newton-Verfahren implementiert und das auf Taschenrechnern ausgeführt wird. Die Frage lautet: Kann die KI für allgemeines intelligentes Verhalten das leisten, was Newton für die Quadratwurzel getan hat? Wir glauben, die Antwort ist „ja“.

Im weiteren Verlauf dieses Abschnitts skizzieren wir vier grundlegende Arten von Agentenprogrammen, die die Prinzipien verkörpern, auf denen fast alle intelligenten Systeme aufbauen:

- Einfache Reflexagenten
- Modellbasierte Reflexagenten
- Zielbasierte Agenten
- Nutzenbasierte Agenten

Jede Art von Agentenprogramm kombiniert spezielle Komponenten auf bestimmte Weise, um Aktionen zu erzeugen. Abschnitt 2.4.6 erklärt in allgemeiner Form, wie man all diese Agenten in *lernende Agenten* umwandelt, die die Leistung ihrer Komponenten verbessern können, um bessere Aktionen zu generieren. Schließlich wird in Abschnitt 2.4.7 die Vielfalt der Möglichkeiten beschrieben, wie die Komponenten selbst innerhalb des Agenten dargestellt werden können. Diese Vielfalt stellt ein wichtiges Ordnungsprinzip sowohl für das Fachgebiet als auch für dieses Buch dar.

2.4.2 Einfache Reflexagenten

Die simpelste Art von Agenten sind die **einfachen Reflexagenten**. Diese Agenten wählen Aktionen auf der Basis des *aktuellen* Perzepts aus und ignorieren den Rest der Perzepthistorie. Zum Beispiel ist der Staubsauger-Agent, dessen Agentenfunktion in ► Abbildung 2.3 tabellarisch dargestellt ist, ein einfacher Reflexagent, weil seine Entscheidung nur auf der aktuellen Position und darauf basiert, ob dieses Feld Schmutz enthält. Ein Agentenprogramm für diesen Agenten ist in ► Abbildung 2.8 dargestellt.

```
function REFLEX-VACUUM-AGENT([location,status]) returns eine Aktion

  if status = Schmutzig then return Saugen
  else if location = A then return Rechts
  else if location = B then return Links
```

Abbildung 2.8: Das Agentenprogramm für einen einfachen Reflexagenten in der Staubsauger-Umgebung mit zwei Feldern. Dieses Programm implementiert die Agentenfunktion aus Abbildung 2.3.

Beachten Sie, dass das Programm des Staubsauger-Agenten im Vergleich zur entsprechenden Tabelle tatsächlich sehr klein ist. Die offensichtlichste Reduzierung ergibt sich daraus, dass die Perzepthistorie ignoriert wird, wodurch sich die Anzahl der relevanten Perzeptfolgen von 4^T auf nur 4 verringert. Eine weitere, kleine Reduktion ergibt sich aus der Tatsache, dass, falls das aktuelle Feld schmutzig ist, die zu wählende Aktion nicht von der Position des Agenten abhängt. Obwohl wir das Agentenprogramm mit **if-then-else**-Anweisungen geschrieben haben, ist es derartig einfach, dass es auch als boolescher Schaltkreis implementiert werden kann.

Einfache reflexartige Verhaltensweisen treten auch in komplexeren Umgebungen auf. Stellen Sie sich vor, Sie wären der Fahrer des automatisierten Taxis. Wenn das vorausfahrende Auto bremst und seine Bremslichter aufleuchten, dann sollten Sie dies erkennen und ebenfalls den Bremsvorgang einleiten. Mit anderen Worten: Es findet eine Verarbeitung der visuellen Eingabe statt, um die Bedingung zu schaffen, die wir „das Auto vor uns bremst“ nennen. Dies löst dann eine bestehende Verbindung im Agentenprogramm zur Aktion „Bremsvorgang einleiten“ aus. Wir nennen eine solche Verbindung eine **Bedingung-Aktion-Regel**[6], geschrieben als:

if *Auto-vor-uns-bremst* **then** *Bremsvorgang-einleiten*

Auch Menschen haben viele solcher Verbindungen, von denen einige erlernte Reaktionen sind (wie beim Autofahren) und einige angeborene Reflexe (wie das Blinzeln, wenn sich etwas dem Auge nähert). Im Rahmen dieses Buchs zeigen wir verschiedene Möglichkeiten auf, wie solche Verbindungen erlernt und implementiert werden können.

Das Programm in ► Abbildung 2.8 ist spezifisch für eine bestimmte Staubsauger-Umgebung. Ein allgemeinerer und flexiblerer Ansatz besteht darin, zunächst einen universellen Interpreter für Bedingung-Aktion-Regeln zu erstellen und dann Regelsätze für spezifische Aufgabenumgebungen festzulegen. ► Abbildung 2.9 gibt die Struktur dieses allgemeinen Programms in schematischer Form wieder und zeigt, wie es die Bedingung-Aktion-Regeln dem Agen-

6 Auch **Situation-Aktion-Regel**, **Produktions-** oder **if-then-Regel**. Das Besondere an diesen Regeln ist, dass sie immer eine Aktion „produzieren“. Der Begriff **if-then**-Regel wird auch allgemeiner verwendet.

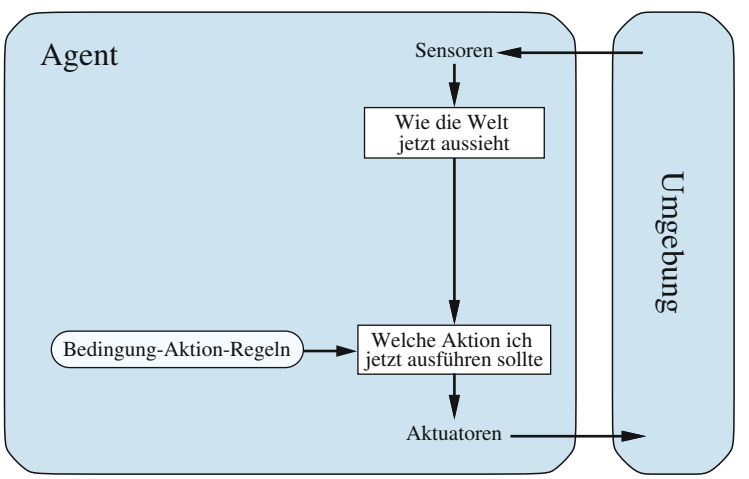


Abbildung 2.9: Schematische Darstellung eines einfachen Reflexagenten. Wie verwenden Rechtecke, um den aktuellen internen Zustand des Entscheidungsprozesses im Agenten zu kennzeichnen, und Ovale, um die Hintergrundinformationen darzustellen, die in dem Prozess benutzt werden.

```
function SIMPLE-REFLEX-AGENT(percept) returns eine Aktion
  persistent: rules, eine Menge von Bedingung-Aktion-Regeln

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

Abbildung 2.10: Ein einfacher Reflexagent. Er handelt nach einer Regel, deren Bedingung mit dem durch das Perzept definierten aktuellen Zustand übereinstimmt.

ten erlauben, die Verbindung vom Perzept zur Aktion herzustellen. Machen Sie sich keine Sorgen, wenn dies trivial erscheint; es wird in Kürze interessanter.

Ein Agentenprogramm für ► Abbildung 2.9 ist in ► Abbildung 2.10 dargestellt. Die Funktion INTERPRET-INPUT erzeugt aus dem Perzept eine abstrahierte Beschreibung des aktuellen Zustands und die Funktion RULE-MATCH gibt die erste Regel aus dem Regelsatz zurück, die mit der gegebenen Zustandsbeschreibung übereinstimmt. Beachten Sie, dass die Beschreibung hinsichtlich „Regeln“ und „Übereinstimmung“ rein konzeptuell ist; wie oben erwähnt, können die tatsächlichen Implementierungen so einfach sein wie eine Menge von Logikgattern, die einen booleschen Schaltkreis implementieren. Alternativ kann auch ein „neuronaler“ Schaltkreis verwendet werden, bei dem die logischen Gatter durch die nichtlinearen Einheiten künstlicher neuronaler Netze ersetzt werden (siehe Kapitel 21).

Einfache Reflexagenten haben die großartige Eigenschaft, dass sie einfach sind. Allerdings sind sie auch nur von begrenzter Intelligenz. Der Agent in ► Abbildung 2.10 funktioniert *nur dann, wenn die richtige Entscheidung allein auf Grundlage des aktuellen Perzepts getroffen werden kann – das heißt nur, wenn die Umgebung vollständig beobachtbar ist.*

Selbst das kleinste bisschen Unbeobachtbarkeit kann zu ernsthaften Problemen führen. Die oben beschriebene Bremsregel geht zum Beispiel davon aus, dass die Bedingung *Auto-vor-uns-bremst* aus dem aktuellen Perzept – einem einzigen Videobild – bestimmt werden kann. Dies funktioniert, wenn das vorausfahrende Fahrzeug ein mittig angebrachtes (und damit eindeutig identifizierbares) Bremslicht hat. Leider haben ältere Modelle unterschiedliche Anordnungen von Rücklicht, Bremslicht und Blinker und es ist nicht immer möglich, anhand eines einzigen Bilds zu erkennen, ob das Auto bremst oder nur die Rücklichter an hat. Ein einfacher Reflexagent, der hinter einem solchen Auto fährt, würde entweder ständig und unnötig bremsen oder – noch schlimmer – überhaupt nicht bremsen.

Ein ähnliches Problem kann uns in der Staubsaugerwelt begegnen. Nehmen wir an, ein einfacher Staubsauger-Reflexagent verliert seinen Positionssensor und besitzt nur noch einen Schmutzsensor. Ein solcher Agent hat lediglich zwei mögliche Perzepte: [*Schmutzig*] und [*Sauber*]. Er kann *Saugen* als Reaktion auf [*Schmutzig*] ausführen – doch was sollte er als Reaktion auf [*Sauber*] tun? Die *Links*-Bewegung scheitert (immer), falls er in Feld *A* startet, und die *Rechts*-Bewegung scheitert (immer), falls er in Feld *B* startet. Endlosschleifen sind für einfache Reflexagenten in teilweise beobachtbaren Umgebungen oft unvermeidbar.

Ein Austritt aus Endlosschleifen ist möglich, wenn der Agent seine Aktionen **zufällig auswählen** (randomisieren) kann. Empfängt der Staubsauger-Agent zum Beispiel [*Sauber*], dann könnte er eine Münze werfen, um zwischen *Rechts* und *Links* zu wählen. Man kann leicht zeigen, dass der Agent das andere Feld in durchschnittlich zwei Schritten erreicht. Wenn dieses Feld dann schmutzig ist, wird der Agent es säubern und die Aufgabe ist erledigt. Ein einfacher Reflexagent mit Zufallsauswahl könnte somit einem deterministischen einfachen Reflexagenten überlegen sein.

In Abschnitt 2.3 haben wir erwähnt, dass zufälliges Verhalten der richtigen Art in einigen Multiagentenumgebungen rational sein kann. In Einzelagentenumgebungen ist die Randomisierung normalerweise nicht rational. Es ist ein nützlicher Trick, der einem einfachen Reflexagenten in manchen Situationen hilft, aber in den meisten Fällen können wir mit ausgefeilteren deterministischen Agenten viel mehr erreichen.

2.4.3 Modellbasierte Reflexagenten

Der effektivste Weg für einen Agenten, mit teilweiser Beobachtbarkeit umzugehen, ist, *über den aktuell nicht sichtbaren Teil der Welt quasi Buch zu führen*. Das heißt, der Agent sollte eine Art **internen Zustand** verwalten, der von der Perzepthistorie abhängt und dadurch zumindest einige der unbeobachteten Aspekte des aktuellen Zustands widerspiegelt. Für das Bremsproblem ist der interne Zustand nicht allzu umfangreich – es ist nur das vorhergehende Kamerabild, sodass der Agent erkennen kann, wenn zwei rote Lichter an den Seiten des Fahrzeugs gleichzeitig an- oder ausgehen. Für andere Fahraufgaben, wie z. B. den Spurwechsel, muss sich der Agent merken, wo sich die anderen Autos befinden, falls er sie nicht alle gleichzeitig sehen kann. Und damit überhaupt eine Fahrt möglich ist, muss sich der Agent merken, wo seine Schlüssel sind.

Um diese internen Zustandsinformationen über die Zeit zu aktualisieren, müssen zwei Arten von Wissen in irgendeiner Form im Agentenprogramm codiert werden. Erstens benötigen wir Informationen darüber, wie sich die Welt im Laufe der Zeit verändert, wobei man im Wesentlichen zwei Teile unterscheiden kann: die Auswirkungen der Aktionen des Agenten und die Entwicklung der Welt unabhängig vom Agenten. Wenn der Agent zum Beispiel das Lenkrad im Uhrzeigersinn bewegt, dann dreht sich das Auto nach rechts, und wenn es regnet, können die Kameras des Autos nass werden. Dieses Wissen darüber, „wie die Welt funktioniert“ – ob in einfachen booleschen Schaltkreisen oder in vollständigen wissenschaftlichen Theorien implementiert –, wird als **Transitionsmodell** der Welt bezeichnet.

Zweitens benötigen wir Informationen darüber, wie sich der Zustand der Welt in den Perzepten des Agenten widerspiegelt. Wenn z. B. das vorausfahrende Auto einen Bremsvorgang einleitet, so erscheinen ein oder mehrere rot beleuchtete Bereiche im nach vorne gerichteten Kamerabild, und wenn die Kamera nass wird, erscheinen tropfenförmige Objekte im Bild, die die Straße teilweise verdecken. Diese Art von Wissen wird als **Sensormodell** bezeichnet.

Zusammen ermöglichen das Transitionsmodell und das Sensormodell einem Agenten, den Zustand der Welt zu verfolgen – soweit dies angesichts der Einschränkungen durch die Sensoren des Agenten möglich ist. Ein Agent, der solche Modelle verwendet, wird als **modellbasierter Agent** bezeichnet.

► Abbildung 2.11 zeigt die Struktur des modellbasierten Reflexagenten mit internem Zustand und verdeutlicht, wie das aktuelle Perzept mit dem alten internen Zustand verknüpft wird, um die aktualisierte Beschreibung des aktuellen Zustands zu generieren, basierend auf dem Modell des Agenten über Funktionsweise der Welt. Das Agentenprogramm ist in ► Abbildung 2.12. dargestellt. Der interessante Teil ist die Funktion UPDATE-STATE, die für die Erstellung der neuen internen Zustandsbeschreibung verantwortlich ist. Wie Modelle und Zustände im Detail dargestellt werden, variiert stark und hängt von der Art der Umgebung und der speziellen Technologie ab, die beim Entwurf des Agenten verwendet wird.

Unabhängig von der Art der Darstellung ist es für den Agenten selten möglich, den aktuellen Zustand einer teilweise beobachtbaren Umgebung *genau* zu bestimmen. Stattdessen repräsentiert das Kästchen mit der Aufschrift „Wie die Welt jetzt aussieht“ (► Abbildung 2.11) die „beste Vermutung“ des Agenten (oder manchmal beste Vermutungen, wenn der Agent mehrere Möglichkeiten in Betracht zieht). Zum Beispiel kann ein automatisiertes Taxi möglicherweise nicht um den großen Lkw herumsehen, der vor ihm angehalten hat, und kann

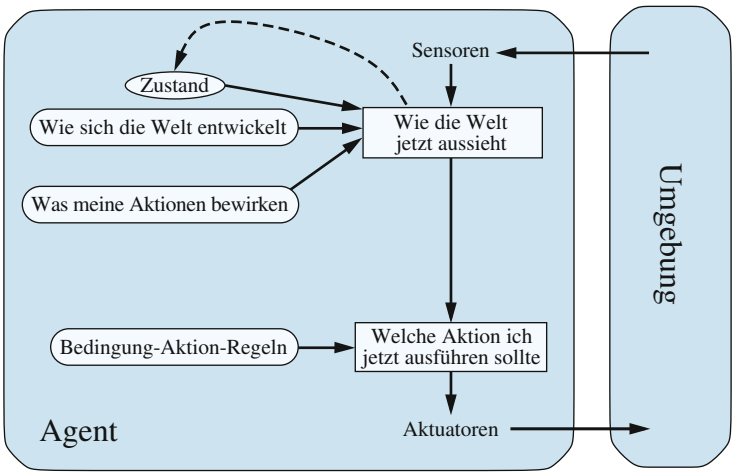


Abbildung 2.11: Ein modellbasierter Reflexagent.

function MODEL-BASED-REFLEX-AGENT(*percept*) **returns** eine Aktion
  persistent: *state*, die aktuelle Auffassung des Agenten vom Zustand der Welt
    transition_model, eine Beschreibung, wie der nächste Zustand vom aktuellen Zustand und der Aktion abhängt
    sensor_model, eine Beschreibung, wie sich der aktuelle Zustand der Welt in den Perzepten des Agenten widerspiegelt
    rules, eine Menge von Bedingung-Aktion-Regeln
    action, die vorherige Aktion, anfangs keine

  state ← UPDATE-STATE(*state*, *action*, *percept*, *transition_model*, *sensor_model*)
  rule ← RULE-MATCH(*state*, *rules*)
  action ← *rule*.ACTION
  return *action*

Abbildung 2.12: Ein modellbasierter Reflexagent. Er verwaltet den aktuellen Zustand der Welt mithilfe eines internen Modells. Dann wählt er eine Aktion auf die gleiche Weise wie der Reflexagent aus.

nur raten, was die Ursache für den Stau sein könnte. Die Unsicherheit über den aktuellen Zustand ist also eventuell unvermeidlich, aber der Agent muss trotzdem eine Entscheidung treffen.

2.4.4 Zielbasierte Agenten

Es reicht nicht immer aus, etwas über den aktuellen Zustand der Umgebung zu wissen, um zu entscheiden, was zu tun ist. Zum Beispiel kann das Taxi an einer Straßenkreuzung links abbiegen, rechts abbiegen oder geradeaus weiterfahren. Die richtige Entscheidung hängt davon ab, wohin das Taxi fahren will. Mit anderen Worten: Der Agent benötigt neben einer Beschreibung des aktuellen Zustands auch gewisse Informationen über das **Ziel**, das wünschenswerte Situationen beschreibt – z. B. an einem bestimmten Ort angekommen zu sein. Das Agentenprogramm kann dies mit dem Modell kombinieren (dieselben Informationen, die im modellbasierten Reflexagenten verwendet wurden), um Aktionen auszuwählen, die das Ziel erreichen. ► Abbildung 2.13 zeigt die Struktur des zielbasierten Agenten.

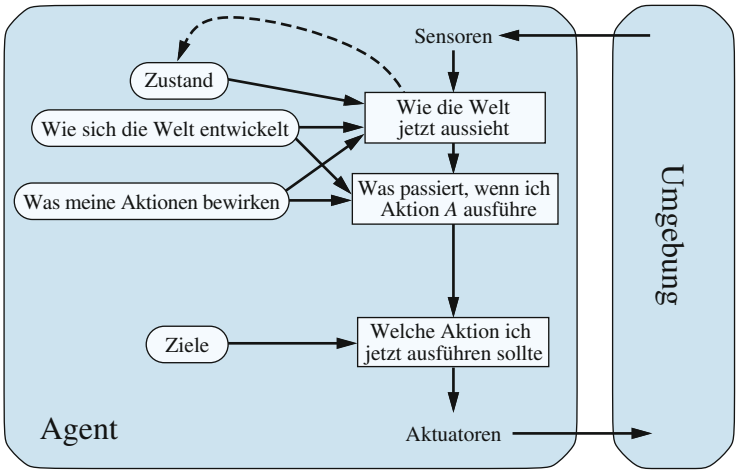


Abbildung 2.13: Ein modellbasierter, zielbasierter Agent. Er verwaltet den Zustand der Welt sowie eine Reihe von Zielen, die er zu erreichen versucht, und wählt eine Aktion aus, die (letztendlich) zum Erreichen seiner Ziele führt.

Manchmal ist die Auswahl von zielbasierten Aktionen einfach – zum Beispiel, wenn das Ziel direkt mit einer einzigen Aktion erreicht wird. Manchmal ist es etwas schwieriger – zum Beispiel, wenn der Agent lange Folgen komplizierter Aktionen in Betracht ziehen muss, um das Ziel zu erreichen. **Suchen** (Kapitel 3 bis 5) und **Planen** (Kapitel 11) sind die Teilgebiete der KI, die sich damit beschäftigen, Aktionsfolgen zu finden, die die Ziele des Agenten erreichen.

Beachten Sie, dass sich diese Art der Entscheidungsfindung grundlegend von den zuvor beschriebenen Bedingung-Aktion-Regeln unterscheidet, da hier auch die Zukunft berücksichtigt wird – sowohl „Was wird passieren, wenn ich dies und jenes tue?“ als auch „Wird mich das glücklich machen?“. In den Entwürfen der Reflexagenten wird diese Information nicht explizit dargestellt, da die eingebauten Regeln direkt von Perzepten auf Aktionen abbilden. Der Reflexagent bremst, wenn er Bremslichter sieht, Punkt. Er hat keine Ahnung, warum. Ein zielbasierter Agent bremst, wenn er Bremslichter sieht, weil er vorhersieht, das dies die einzige Aktion ist, die ihm hilft, sein Ziel, nicht mit anderen Autos zusammenzustoßen, zu erreichen.

Obwohl der zielbasierte Agent weniger effizient erscheint, ist er flexibler, weil das Wissen, das seine Entscheidungen unterstützt, explizit dargestellt wird und verändert werden kann. Zum Beispiel kann das Verhalten eines zielbasierten Agenten leicht dahingehend geändert werden, ein anderes Fahrtziel anzusteuern, indem einfach diese neue Position als Ziel angegeben wird. Die Regeln des Reflexagenten, wann er abbiegen und wann er geradeaus fahren soll, funktionieren nur für einen einzigen Zielort; sie müssen alle ersetzt werden, wenn er irgendwo anders hinfahren soll.

2.4.5 Nutzenbasierte Agenten

Ziele allein reichen in den meisten Umgebungen nicht aus, um qualitativ hochwertiges Verhalten zu erzeugen. Zum Beispiel werden viele Aktionsfolgen das Taxi an seinen Zielort bringen (und damit das Ziel des Agenten erreichen), aber einige sind schneller, sicherer, zuverlässiger oder billiger als andere. Ziele bieten nur eine grobe binäre Unterscheidung zwischen „glücklichen“ und „unglücklichen“ Zuständen. Ein allgemeineres Leistungsmaß sollte einen Vergleich verschiedener Zustände der Welt danach erlauben, *wie* glücklich sie den Agenten

machen würden. Da „glücklich" nicht sehr wissenschaftlich klingt, verwenden Wirtschaftswissenschaftler und Informatiker stattdessen den Begriff **Nutzen** (*utility*).

Wir haben bereits gesehen, dass ein Leistungsmaß jeder gegebenen Folge von Umgebungszuständen eine Punktzahl zuweist, sodass man leicht zwischen mehr und weniger wünschenswerten Wegen zum Zielort des Taxis unterscheiden kann. Die **Nutzenfunktion** eines Agenten ist im Wesentlichen eine Internalisierung des Leistungsmaßes. Unter der Voraussetzung, dass die interne Nutzenfunktion und das externe Leistungsmaß übereinstimmen, ist ein Agent, der Aktionen auswählt, um seinen Nutzen zu maximieren, hinsichtlich des externen Leistungsmaßes rational.

Es sei noch einmal betont, dass dies nicht die *einzige* Möglichkeit ist, rational zu sein – wir haben bereits ein rationales Agentenprogramm für die Staubsaugerwelt kennengelernt (► Abbildung 2.8), das nichts von seiner Nutzenfunktion weiß –, doch wie zielbasierte Agenten bietet ein nutzenbasierter Agent viele Vorteile in Bezug auf Flexibilität und Lernen. Darüber hinaus sind Ziele in zwei Arten von Fällen unzureichend, wo aber ein nutzenbasierter Agent trotzdem rationale Entscheidungen treffen kann. Der erste Fall ist, wenn es widersprüchliche Ziele gibt, von denen nur einige erreicht werden können (z. B. hohe Geschwindigkeit und volle Sicherheit). Hier gibt die Nutzenfunktion die angemessene Abwägung vor. Der zweite Fall ist, wenn es mehrere Ziele gibt, die der Agent anstreben kann, doch von denen keines mit Sicherheit erreicht werden kann. Dann bietet der Nutzen eine Möglichkeit, die Erfolgswahrscheinlichkeit gegen die Wichtigkeit der Ziele abzuwägen.

Teilweise Beobachtbarkeit und Nichtdeterminismus sind in der realen Welt allgegenwärtig – und damit auch die Entscheidungsfindung unter Unsicherheit. Technisch gesehen wählt ein rationaler nutzenbasierter Agent die Aktion aus, die den **erwarteten Nutzen** der Aktionsergebnisse maximiert – d. h. den Nutzen, den der Agent im Durchschnitt erwartet, wenn er die Wahrscheinlichkeiten und Nutzenwerte der einzelnen Ergebnisse berücksichtigt. (Anhang A definiert den Erwartungswert genauer.) In Kapitel 16 zeigen wir, dass sich jeder rationale Agent so verhalten muss, *als ob* er eine Nutzenfunktion besäße, deren Erwartungswert er zu maximieren versucht. Ein Agent, der eine *explizite* Nutzenfunktion besitzt, kann rationale Entscheidungen mit einem allgemeinen Algorithmus treffen, der nicht von der spezifischen Nutzenfunktion abhängt, die er maximiert. Auf diese Weise wird die „globale" Definition von Rationalität – in der die Agentenfunktionen mit der höchsten Leistung als rational bezeichnet werden – in eine „lokale" Einschränkung für den Entwurf rationaler Agenten umgewandelt, die in einem einfachen Programm ausgedrückt werden kann.

Die Struktur eines nutzenbasierten Agenten ist in ► Abbildung 2.14 dargestellt. Nutzenbasierte Agentenprogramme tauchen in Kapitel 16 und 17 wieder auf, wo wir entscheidungsfähige Agenten entwerfen, die mit der Unsicherheit umgehen müssen, die in nichtdeterministischen oder teilweise beobachtbaren Umgebungen herrscht. Entscheidungsfindung in Multiagentenumgebungen wird in Kapitel 18, ebenfalls im Rahmen der Nutzentheorie, untersucht.

An dieser Stelle fragt sich der Leser vielleicht: „Ist es wirklich so simpel? Wir erstellen einfach Agenten, die den erwarteten Nutzen maximieren, und schon sind wir fertig?" Es stimmt, solche Agenten wären intelligent, aber einfach ist es nicht. Ein nutzenbasierter Agent muss seine Umgebung modellieren und verfolgen – Aufgaben, für die umfangreiche Forschung in den Bereichen Wahrnehmung, Repräsentation, Schlussfolgerung und Lernen betrieben wurde. Die Ergebnisse dieser Forschung füllen viele Kapitel dieses Buchs. Die Wahl der nutzenmaximierenden Aktionskette ist ebenfalls eine schwierige Aufgabe, die ausgeklügelte Algorithmen erfordert, womit wir mehrere weitere Kapitel füllen. Selbst mit diesen Algorithmen ist perfekte Rationalität normalerweise in der Praxis aufgrund der Komplexität der Berechnungen unerreichbar, wie wir in Kapitel 1 festgestellt haben. Wir wollen nicht verschweigen, dass nicht alle nutzenbasierten Agenten modellbasiert sind; wir werden in Kapitel 22 und 26 sehen, dass ein **modellfreier Agent** lernen kann, welche Aktion in einer bestimmten Situation am besten ist, ohne jemals genau zu lernen, wie diese Aktion die Umgebung verändert.

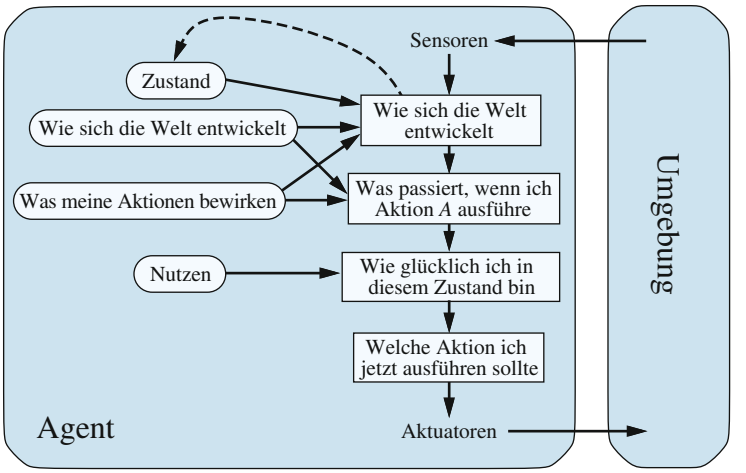


Abbildung 2.14: Ein modellbasierter, nutzenbasierter Agent. Er verwendet ein Modell der Welt zusammen mit einer Nutzenfunktion, die seine Präferenzen hinsichtlich der Zustände der Welt bewertet. Dann wählt er die Aktion, die zum besten erwarteten Nutzen führt, wobei der erwartete Nutzen berechnet wird, indem der Mittelwert über alle möglichen Ergebniszustände, gewichtet nach der Wahrscheinlichkeit des Ergebnisses, gebildet wird.

Schließlich wird bei all dem davon ausgegangen, dass der Entwickler die Nutzenfunktion korrekt spezifizieren kann; in den Kapiteln 17, 18 und 22 wird das Thema der unbekannten Nutzenfunktionen ausführlicher behandelt.

2.4.6 Lernende Agenten

Wir haben Agentenprogramme beschrieben, die Aktionen nach verschiedenen Methoden auswählen. Doch wir haben bisher nicht erklärt, wie die Agentenprogramme *entstehen*. In seiner berühmten frühen Arbeit erwägt Turing (1950) die Idee, seine intelligenten Maschinen tatsächlich von Hand zu programmieren. Er schätzt, wie viel Arbeit dies bedeuten würde, und kommt zu dem Schluss: „Eine schnellere Methode scheint wünschenswert." Turing schlägt als Methode vor, lernende Maschinen zu erstellen und diese dann zu trainieren. In vielen Bereichen der Künstlichen Intelligenz ist dies auch mittlerweile die bevorzugte Methode, um moderne Systeme zu entwickeln. Jeder Agententyp (modellbasiert, zielbasiert, nutzenbasiert usw.) kann als lernender Agent gebaut werden (oder eben nicht).

Wie bereits erwähnt, hat Lernen noch einen weiteren Vorteil: Lernen ermöglicht es dem Agenten, in anfänglich unbekannten Umgebungen zu arbeiten und mit der Zeit kompetenter zu werden, als es sein anfängliches Wissen allein erlauben würde. In diesem Abschnitt stellen wir kurz die wichtigsten Ideen von lernenden Agenten vor. Im weiteren Verlauf des Buchs gehen wir auf die Möglichkeiten und Methoden des Lernens für bestimmte Arten von Agenten ein. In den Kapiteln 19–22 werden die Lernalgorithmen selbst näher beleuchtet.

Ein lernender Agent kann in vier konzeptionelle Komponenten unterteilt werden, wie in ► Abbildung 2.15 gezeigt. Die wichtigste Unterscheidung ist die zwischen dem **Lernelement**, das für Verbesserungen zuständig ist, und dem **Leistungselement**, das für die Auswahl externer Aktionen verantwortlich ist. Das Leistungselement ist das, was wir zuvor als den ganzen Agent betrachtet haben: Es empfängt Perzepte und entscheidet über Aktionen. Das Lernelement nutzt das Feedback der **Kritik**-Komponente zur Leistung des Agenten und bestimmt, wie das Leistungselement modifiziert werden sollte, um in Zukunft besser abzuschneiden.

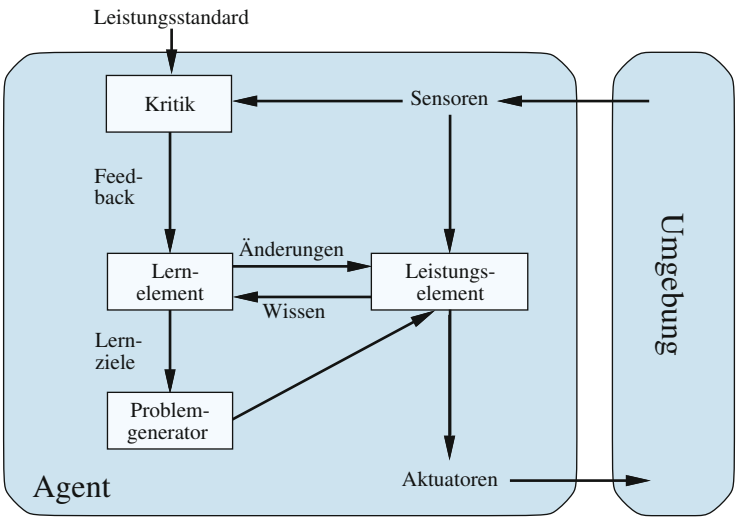


Abbildung 2.15: Ein allgemeiner lernender Agent. Der Kasten „Leistungselement" repräsentiet das, was wir bisher als das gesamte Agentenprogramm betrachtet haben. Jetzt wird der Kasten „Lernelement" dazu verwendet, dieses Programm zu modifizieren, um seine Leistung zu verbessern.

Der Entwurf des Lernelements hängt sehr stark vom Entwurf des Leistungselements ab. Wenn man versucht, einen Agenten zu entwerfen, der eine bestimmte Fähigkeit erlernt, lautet die erste Frage nicht: „Wie bringe ich ihn dazu, dies zu lernen?", sondern: „Welche Art von Leistungselement wird mein Agent verwenden, um dies zu tun, sobald er gelernt hat, wie es geht?" Für einen gegebenen Entwurf des Leistungselements können Lernmechanismen konstruiert werden, um jeden Teil des Agenten zu verbessern.

Die Kritik teilt dem Lernelement mit, wie gut der Agent in Bezug auf einen festgelegten Leistungsstandard abschneidet. Die Kritik ist notwendig, weil die Perzepte selbst keinen Hinweis auf den Erfolg des Agenten liefern. Zum Beispiel könnte ein Schachprogramm ein Perzept erhalten, das anzeigt, dass das Programm seinen Gegner schachmatt gesetzt hat, aber es benötigt einen Leistungsstandard, um zu wissen, dass dies eine gute Sache ist; das Perzept selbst sagt das nicht. Es ist wichtig, dass der Leistungsstandard festgelegt ist. Konzeptionell sollte man ihn sich als außerhalb des Agenten befindlich vorstellen, weil der Agent ihn nicht verändern darf, um den Standard an sein eigenes Verhalten anzupassen.

Die letzte Komponente des lernenden Agenten ist der **Problemgenerator**. Er ist dafür verantwortlich, Aktionen vorzuschlagen, die zu neuen und informativen Erfahrungen führen. Wenn es nur nach dem Leistungselement ginge, so würde es immer weiter die Aktionen ausführen, die seines Wissens nach am besten sind, doch wenn der Agent bereit ist, sich ein wenig auszuprobieren und kurzfristig einige möglicherweise suboptimale Aktionen auszuführen, dann könnte er auf lange Sicht wesentlich bessere Aktionen entdecken. Die Aufgabe des Problemgenerators ist es, diese Erkundungsaktionen vorzuschlagen. So verhalten sich Wissenschaftler, wenn sie Experimente durchführen. Galileo dachte nicht, dass der Akt, Steine von einem Turm in Pisa fallen zu lassen, an sich wertvoll sei. Es ging ihm nicht darum, die Steine zu zertrümmern oder die Gehirne der unglücklichen Passanten zu modifizieren. Sein Ziel war es, sein eigenes Gehirn zu modifizieren, indem er eine bessere Theorie für die Bewegung von Objekten ausfindig machte.

Das Lernelement kann Änderungen an jeder der in den Agentendiagrammen (► Abbildungen 2.9, 2.11, 2.13 und 2.14) dargestellten „Wissens"-Komponenten vornehmen. In den ein-

fachsten Fällen wird direkt aus der Perzeptfolge gelernt. Durch die Beobachtung von Paaren aufeinanderfolgender Zustände der Umgebung kann der Agent lernen, „was meine Aktionen bewirken“ und „wie sich die Welt entwickelt“ in Reaktion auf seine Aktion. Wenn das automatisierte Taxi z. B. bei einer Fahrt auf nasser Straße einen bestimmten Bremsdruck ausübt, wird es bald herausfinden, wie viel Bremsleistung dadurch tatsächlich erreicht wird und ob es von der Straße abrutscht. Der Problemgenerator könnte bestimmte Teile des Modells identifizieren, die verbesserungsbedürftig sind, und Experimente vorschlagen, wie z. B. das Ausprobieren der Bremsen auf verschiedenen Straßenbelägen unter unterschiedlichen Bedingungen.

Die Modellkomponenten eines modellbasierten Agenten so zu verbessern, dass sie mehr mit der Realität übereinstimmen, ist fast immer eine gute Idee, unabhängig vom externen Leistungsstandard. (In manchen Fällen ist es aus rechnerischer Sicht besser, ein einfaches, aber leicht ungenaues Modell zu haben, als ein perfektes, aber höllisch komplexes Modell.) Informationen aus dem externen Standard werden benötigt, wenn man versucht, eine Reflexkomponente oder eine Nutzenfunktion zu lernen.

Nehmen wir zum Beispiel an, der Taxifahrer-Agent erhält kein Trinkgeld von Fahrgästen, die während der Fahrt gründlich durchgeschüttelt wurden. Der externe Leistungsstandard muss den Agenten darüber informieren, dass sich der Verlust von Trinkgeld negativ auf seine Gesamtleistung auswirkt; so könnte der Agent lernen, dass aggressive Manöver nicht zu seinem eigenen Nutzen beitragen. In gewissem Sinne identifiziert der Leistungsstandard einen Teil des eingehenden Perzepts als eine **Belohnung** (bwz. **Strafe**), wodurch der Agent ein direktes Feedback zur Qualität seines Verhaltens bekommt. Fest „verdrahtete“ Leistungsstandards wie Schmerz und Hunger bei Tieren können auf diese Weise verstanden werden.

Allgemeiner gesagt, können *menschliche Entscheidungen* Informationen über menschliche Vorlieben liefern. Nehmen wir zum Beispiel an, das Taxi weiß nicht, dass Menschen gewöhnlich keine lauten Geräusche mögen, und kommt auf die Idee, ständig zu hupen, um sicherzustellen, dass alle Fußgänger mitbekommen, dass sich das Taxi nähert. Das darauf folgende Verhalten der Menschen – sich die Ohren zuhalten, Schimpfwörter benutzen und möglicherweise die Kabel zur Hupe durchschneiden – würde dem Agenten Hinweise bieten, mit denen er seine Nutzenfunktion aktualisieren kann. Dieses Thema wird in Kapitel 22 weiter diskutiert.

Zusammenfassend halten wir fest: Agenten besitzen eine Vielzahl von Komponenten, die auf viele Arten im Agentenprogramm dargestellt werden können, sodass es eine große Bandbreite an Lernmethoden zu geben scheint. Es gibt jedoch auch einen verbindenden Aspekt: Lernen in intelligenten Agenten lässt sich als ein Prozess zusammenfassen, der jede Komponente des Agenten verändert, um die Komponenten enger mit den verfügbaren Feedback-Informationen abzustimmen und dadurch die Gesamtleistung des Agenten zu verbessern.

2.4.7 Wie die Komponenten von Agentenprogrammen funktionieren

Laut unserer bisherigen Beschreibung (auf sehr hohem Niveau) bestehen Agentenprogramme aus verschiedenen Komponenten, deren Aufgabe es ist, Fragen zu beantworten wie: „Wie sieht die Welt jetzt aus?“, „Welche Aktion sollte ich jetzt ausführen?“ oder „Was bewirken meine Aktionen?“. Und für einen KI-Studenten lautet die nächste Frage: „Wie in aller Welt funktionieren diese Komponenten?“ Man bräuchte ungefähr tausend Seiten, um diese Frage ordentlich zu beantworten, doch wir wollen hier die Aufmerksamkeit des Lesers auf einige grundlegende Unterscheidungen zwischen den verschiedenen Möglichkeiten lenken, wie die Komponenten die Umgebung, in der der Agent lebt, darstellen können.

Grob gesagt können wir die Darstellungen entlang einer Achse zunehmender Komplexität und Ausdrucksstärke einordnen – **atomar**, **faktorisiert** und **strukturiert**. Diese Konzepte las-

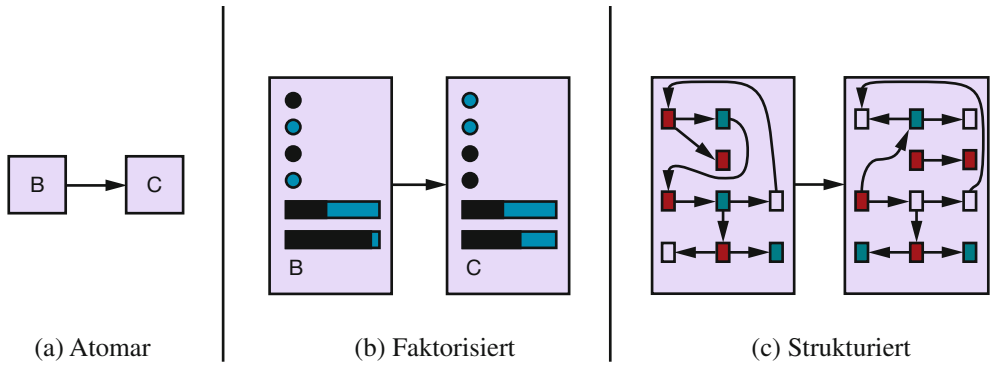


Abbildung 2.16: Drei Möglichkeiten, Zustände und die Übergänge zwischen ihnen darzustellen. (a) Atomare Darstellung: ein Zustand (z. B. B oder C) ist eine Blackbox ohne interne Struktur. (b) Faktorisierte Darstellung: ein Zustand besteht aus einem Vektor von Attributwerten – boolesche Werte, Gleitkommazahlen oder ein fester Satz von Symbolen. (c) Strukturierte Darstellung: Ein Zustand umfasst Objekte, von denen jedes sowohl eigene Attribute als auch Beziehungen zu anderen Objekten haben kann.

sen sich am besten anhand einer bestimmten Agentenkomponente veranschaulichen, z. B. an der, die sich mit „Was meine Aktionen bewirken“ befasst. Diese Komponente beschreibt die Änderungen, die in der Umgebung als Ergebnis einer Aktion auftreten können. ► Abbildung 2.16 zeigt schematische Darstellungen dieser Übergänge.

In einer **atomaren Darstellung** ist jeder Zustand der Welt unteilbar – er hat keine interne Struktur. Betrachten Sie die Aufgabe, eine Fahrtroute von einem Ende eines Lands zum anderen über eine Reihe von Städten zu finden (wir behandeln dieses Problem in ► Abbildung 3.1 auf S. 93). Für die Lösung dieses Problems kann es ausreichen, den Zustand der Welt auf den Namen der Stadt zu reduzieren, in der wir uns befinden – ein einzelnes Wissensatom, eine „Blackbox“ mit nur einer einzigen erkennbaren Eigenschaft: Sie ist entweder mit einer anderen Blackbox identisch oder von ihr verschieden. Die Standardalgorithmen, die der **Suche** und dem **Spielen** (Kapitel 3–5), den **Hidden-Markov-Modellen** (Kapitel 14) und den Markov-Entscheidungsprozessen (Kapitel 17) zugrunde liegen, arbeiten alle mit atomaren Repräsentationen.

Eine **faktorisierte Darstellung** unterteilt jeden Zustand in eine feste Menge von **Variablen** oder **Attributen**, von denen jedes einen **Wert** annehmen kann. Sehen wir uns das oben angegebene Fahrtproblem noch einmal unter einem realitätsnäheren Blickwinkel an, wenn wir mehr als unseren Standort in einer Stadt als atomaren Zustand berücksichtigen müssen: Wir müssen vielleicht darauf achten, wie der Benzinstand in unserem Tank ist, welches unsere aktuellen GPS-Koordinaten sind, ob die Ölwarnleuchte funktioniert, wie viel Geld wir für die Maut haben, welcher Sender im Radio läuft und so weiter. Während zwei verschiedene atomare Zustände nichts gemeinsam haben – sie sind einfach nur unterschiedliche Blackboxes –, können zwei verschiedene faktorisierte Zustände einige gemeinsame Attribute haben (z. B. an einer bestimmten GPS-Position zu sein), andere wiederum nicht (z. B. viel Benzin oder kein Benzin zu haben); auf diese Weise lässt es sich viel einfacher herausfinden, wie man einen Zustand in einen anderen überführen kann. Viele wichtige Bereiche der Künstlichen Intelligenz basieren auf faktorisierten Darstellungen, darunter Algorithmen für Constraint-Satisfaction-Probleme (Probleme unter Rand-/Nebenbedingungen, Kapitel 6), Aussagenlogik (Kapitel 7), Planung (Kapitel 11), Bayes'sche Netze (Kapitel 12–16) sowie verschiedene Algorithmen zum **maschinellen Lernen**.

Für viele Aufgaben ist es wichtig, die Welt in dem Sinne zu begreifen, dass sie nicht nur aus Variablen mit Werten besteht, sondern dass es in ihr *Dinge* gibt, die miteinander *in Beziehung*

stehen. Zum Beispiel könnten wir mitbekommen, dass ein großer Lkw vor uns rückwärts in die Einfahrt eines Milchbauernhofs fährt, doch eine freilaufende Kuh blockiert den Weg des Lkws. Es ist recht unwahrscheinlich, dass in einer faktorisierten Repräsentation bereits ein Attribut *LkwVorausFährtRückwärtsInMilchbauernhofeinfahrtBlockiertVonFreilaufenderKuh* existiert, das den Wert *wahr* oder *falsch* annehmen kann. Stattdessen benötigen wir eine **strukturierte Repräsentation**, in der Objekte wie Kühe und Lkws sowie ihre unterschiedlichen und variierenden Beziehungen explizit beschrieben werden können (► Abbildung 2.16 c). Strukturierte Darstellungen bilden die Basis für relationale Datenbanken und die Prädikatenlogik (Kapitel 8, 9 und 10), für prädikatenlogische Wahrscheinlichkeitsmodelle (Kapitel 15) sowie einen großen Teil der Computerlinguistik (Kapitel 23 und 24). Tatsächlich betrifft fast alles, was Menschen in natürlicher Sprache ausdrücken, Objekte und ihre Beziehungen.

Wie wir bereits erwähnt haben, liegen atomare, faktorisierte und strukturierte Darstellungen auf der Achse der zunehmenden **Ausdrucksstärke**. Grob gesagt kann eine ausdrucksstärkere Darstellung alles, was eine weniger ausdrucksstarke Darstellung erfassen kann, mindestens genauso prägnant wiedergeben, und noch einiges mehr. Häufig ist die ausdrucksstärkere Sprache *wesentlich* prägnanter; zum Beispiel kann man die Regeln des Schachspiels in einer strukturierten Repräsentationssprache wie der Prädikatenlogik auf ein oder zwei Seiten aufschreiben, man benötigt aber Tausende von Seiten, wenn man sie in einer faktorisierten Repräsentationssprache wie der Aussagenlogik verfasst, und etwa 10^{38} Seiten, wenn sie in einer atomaren Sprache wie der der endlichen Automaten geschrieben werden. Andererseits werden Schlussfolgern und Lernen komplexer, wenn die Ausdrucksstärke der Darstellung zunimmt. Um die Vorteile von ausdrucksstarken Repräsentationen auszunutzen und gleichzeitig ihre Nachteile zu vermeiden, müssen intelligente Systeme für die reale Welt möglicherweise an allen Punkten entlang der Achse gleichzeitig arbeiten.

Eine andere Achse für die Darstellung bezieht sich auf das Abbilden von Konzepten auf Orte im physischen Speicher, sei es in einem Computer oder in einem Gehirn. Gibt es eine Eins-zu-eins-Abbildung zwischen Konzepten und Speicherplätzen, so nennen wir das eine **lokalistische Repräsentation**. Wenn hingegen die Darstellung eines Konzepts auf viele Speicherplätze verteilt ist und jeder Speicherort als Teil der Repräsentation mehrerer verschiedener Konzepte verwendet wird, dann ist dies eine **verteilte Repräsentation**. Verteilte Repräsentationen sind robuster gegen Rauschen und Informationsverlust. Bei einer lokalistischen Repräsentation ist die Zuordnung vom Konzept zum Speicherort willkürlich und wenn ein Übertragungsfehler ein paar Bits verstümmelt, könnten wir *Milch* mit dem nicht verwandten Konzept *Mulch* verwechseln. Eine verteilte Repräsentation kann man sich dagegen so vorstellen, dass jedes Konzept einen Punkt im mehrdimensionalen Raum darstellt, und wenn ein paar Bits verstümmelt werden, bewegt man sich zu einem nahe gelegenen Punkt innerhalb dieses Raums, der eine ähnliche Bedeutung hat.

ZUSAMMENFASSUNG

Dieses Kapitel war so etwas wie ein Schnelldurchlauf durch die KI, die wir hier als die Wissenschaft des Agentenentwurfs verstanden haben. Die wichtigsten Punkte, an die Sie sich erinnern sollten, sind:

- Ein **Agent** ist etwas, das in einer Umgebung Perzepte empfängt und dort handelt. Die **Agentenfunktion** spezifiziert die Aktion, die der Agent als Reaktion auf eine beliebige Perzeptfolge ausführt.
- Das **Leistungsmaß** bewertet das Verhalten des Agenten in einer Umgebung. Ein **rationaler Agent** handelt so, dass er auf Grundlage der bisherigen Perzeptfolge den Erwartungswert des Leistungsmaßes maximiert.
- Zu einer **Aufgabenumgebung** gehören das Leistungsmaß, die äußere Umgebung, die Aktuatoren und die Sensoren. Beim Entwurf eines Agenten muss immer der erste Schritt sein, die Aufgabenumgebung so vollständig wie möglich zu spezifizieren.
- Aufgabenumgebungen variieren entlang mehrerer wichtiger Dimensionen. Die Umgebung kann vollständig oder teilweise beobachtbar sein, eine Einzel- oder Multiagentenumgebung sein, deterministisch oder nichtdeterministisch, episodisch oder sequenziell, statisch oder dynamisch, diskret oder stetig und bekannt oder unbekannt sein.
- In den Fällen, in denen das Leistungsmaß unbekannt oder nur schwer korrekt zu spezifizieren ist, besteht ein erhebliches Risiko, dass der Agent das falsche Ziel optimiert. In solchen Fällen sollte der Agentenentwurf die Unsicherheit über das wahre Ziel widerspiegeln.
- Das **Agentenprogramm** implementiert die Agentenfunktion. Es gibt eine Vielzahl grundlegender Entwürfe für Agentenprogramme, die jeweils die Art der Informationen widerspiegeln, die explizit angegeben und im Entscheidungsprozess verwendet werden. Die Entwürfe unterscheiden sich in Effizienz, Kompaktheit und Flexibilität. Der für ein Agentenprogramm geeignete Entwurf hängt von der Art der Umgebung ab.
- **Einfache Reflexagenten** reagieren direkt auf Perzepte, während **modellbasierte Reflexagenten** einen internen Zustand verwalten, um Aspekte der Welt nachzuvollziehen, die im aktuellen Perzept nicht offensichtlich sind. **Zielbasierte Agenten** handeln, um ihre Ziele zu erreichen, und **nutzenbasierte Agenten** versuchen, ihr eigenes erwartetes „Glück“ zu maximieren.
- Alle Agenten können ihre Leistung durch **Lernen** verbessern.

Bibliografische und historische Anmerkungen

Die zentrale Rolle von Aktionen in der Intelligenz – das Konzept des praktischen Schlussfolgerns – geht mindestens bis zur *Nikomachischen Ethik* von Aristoteles zurück. Praktisches Schlussfolgern war auch das Thema von McCarthys einflussreichem Artikel „Programs with Common Sense“ (1958). Die Bereiche Robotik und Kontrolltheorie befassen sich naturgemäß vor allem mit physischen Agenten. Das Konzept eines **Controllers** in der Kontrolltheorie entspricht dem eines Agenten in der KI. Es mag vielleicht überraschen, dass sich die KI den größten Teil ihrer Geschichte mehr auf isolierte Komponenten von Agenten – Frage-Antwort-Systeme, Theorembeweiser, visuelle Systeme und so weiter – als auf ganze Agenten konzentriert hat. Die Diskussion von Agenten im Buch von Genesereth und Nilsson (1987) war eine bemerkenswerte Ausnahme. Die ganzheitliche Sicht auf Agenten ist heute allgemein akzeptiert und zentrales Thema in neueren Büchern (Padgham und Winikoff, 2004; Jones, 2007; Poole und Mackworth, 2017).

In Kapitel 1 wurden die Wurzeln des Rationalitätskonzepts in der Philosophie und der Wirtschaftswissenschaft nachgezeichnet. In der KI war das Konzept anfangs nur von peripherem Interesse, bis es Mitte der 1980er Jahre in vielen Diskussionen über die passenden technischen Grundlagen des Gebiets auftauchte. Jon Doyle sagte in einem Aufsatz (1983) voraus, dass der Entwurf rationaler Agenten zu einer Kernaufgabe der KI werden würde, während andere populäre Themen sich abspalten und neue Disziplinen bilden würden.

Die sorgfältige Beachtung der Eigenschaften einer Umgebung und ihrer Konsequenzen für den Entwurf von rationalen Agenten ist am deutlichsten in der Tradition der Kontrolltheorie zu erkennen – beispielsweise behandeln klassische Kontrollsysteme (Dorf und Bishop, 2004; Kirk, 2004) vollständig beobachtbare, deterministische Umgebungen; stochastische optimale Steuerung (Kumar und Varaiya, 1986; Bertsekas und Shreve, 2007) beschäftigt sich mit teilweise beobachtbaren, stochastischen Umgebungen; und hybride Steuerung (Henzinger und Sastry, 1998; Cassandras und Lygeros, 2006) hat mit Umgebungen zu tun, die sowohl diskrete als auch stetige Elemente enthalten. Die Unterscheidung zwischen vollständig und teilweise beobachtbaren Umgebungen ist ebenfalls zentral in der Literatur zur **dynamischen Programmierung**, die im Bereich des Operations Research entwickelt wurde (Puterman, 1994) und die wir in Kapitel 17 besprechen.

Obwohl einfache Reflexagenten für die behavioristische Psychologie von zentraler Bedeutung waren (siehe Kapitel 1), werden sie von den meisten KI-Forschern als zu einfach angesehen, um einen großen Nutzen zu bringen. (Rosenschein (1985) und Brooks (1986) stellten diese Annahme infrage; siehe Kapitel 26.) Es wurde viel Aufwand für die Suche nach effizienten Algorithmen betrieben, um den Überblick über komplexe Umgebungen zu behalten (Bar-Shalom *et al.*, 2001; Choset *et al.*, 2005; Simon, 2006), das meiste davon im probabilistischen Umfeld.

Zielbasierte Agenten werden überall vorausgesetzt, angefangen bei der aristotelischen Sichtweise des praktischen Schlussfolgerns bis hin zu McCarthys frühen Arbeiten zur logischen KI. Shakey (Fikes und Nilsson, 1971; Nilsson, 1984) war die erste robotische Verkörperung eines logischen, zielbasierten Agenten. Eine vollständige logische Analyse von zielbasierten Agenten erschien in Genesereth und Nilsson (1987) und eine zielbasierte Programmiermethodik, genannt agentenorientierte Programmierung, wurde von Shoham (1993) entwickelt. Der agentenbasierte Ansatz ist heute in der Softwareentwicklung sehr beliebt (Ciancarini und Wooldridge, 2001). Er ist auch in den Bereich der Betriebssysteme eingedrungen, wo sich **Autonomic Computing** auf Computersysteme und Netzwerke bezieht, die sich selbst mit einer Wahrnehmungs- und Handlungsschleife und maschinellen Lernmethoden überwachen und steuern (Kephart und Chess, 2003). Eine Sammlung von Agentenprogrammen, die so entworfen wurde, dass die Programme in einer echten Multiagentenumgebung gut zusammenarbeiten, weist bekanntlich notwendigerweise Modularität auf –- die Programme teilen keinen

internen Zustand und kommunizieren miteinander nur über die Umgebung. Deshalb ist es im Bereich der **Multiagentensysteme** üblich, das Agentenprogramm eines einzelnen Agenten als eine Sammlung von autonomen Subagenten zu entwerfen. In einigen Fällen kann man sogar beweisen, dass das resultierende System die gleichen optimalen Lösungen liefert wie ein monolithischer Entwurf.

Die zielbasierte Sichtweise von Agenten dominiert auch die Tradition der kognitiven Psychologie auf dem Gebiet des Problemlösens, beginnend mit dem enorm einflussreichen Buch *Human Problem Solving* (Newell und Simon, 1972) zieht sie sich durch alle späteren Arbeiten von Newell (Newell, 1990). Ziele, die weiter als Wünsche (allgemein) und Absichten (aktuell verfolgt) analysiert werden, sind zentral für die einflussreiche Theorie der Agenten, die von Michael Bratman (1987) entwickelt wurde.

Wie in Kapitel 1 erwähnt, reicht die Entwicklung der Nutzentheorie als Grundlage für rationales Verhalten Hunderte von Jahren zurück. In der KI hat die frühe Forschung den Nutzen zugunsten von Zielen gemieden, mit einigen Ausnahmen (Feldman und Sproull, 1977). Das Wiederaufleben des Interesses an probabilistischen Methoden in den 1980er Jahren führte zur Akzeptanz der Maximierung des erwarteten Nutzens als allgemeinsten Rahmen für die Entscheidungsfindung (Horvitz *et al.*, 1988). Der Text von Pearl (1988) war der erste in der KI, der die Wahrscheinlichkeits- und Nutzentheorie eingehend behandelte; seine Darstellung praktischer Methoden zum Schlussfolgern und Entscheidungsfindung unter Unsicherheit war wahrscheinlich der größte Einzelfaktor für den schnellen Wechsel zu nutzenbasierten Agenten in den 1990er Jahren (siehe Kapitel 16). Die Formalisierung von Reinforcement Learning innerhalb eines entscheidungstheoretischen Rahmens trug ebenfalls zu dieser Verschiebung bei (Sutton, 1988). Es ist bemerkenswert, dass fast die gesamte KI-Forschung bis vor Kurzem davon ausging, dass das Leistungsmaß exakt und korrekt in Form einer Nutzen- oder Belohnungsfunktion spezifiziert werden kann (Hadfield-Menell *et al.*, 2017a; Russell, 2019).

Der in ► Abbildung 2.15 dargestellte allgemeine Entwurf für lernende Agenten ist ein Klassiker in der Literatur zum maschinellen Lernen (Buchanan *et al.*, 1978; Mitchell, 1997). Beispiele für den Entwurf, wie er in Programmen umgesetzt ist, reichen mindestens bis zu Arthur Samuels (1959, 1967) Lernprogramm für das Damespiel zurück. Lernende Agenten werden in den Kapiteln 19–22 ausführlich behandelt.

Einige frühe Arbeiten zu agentenbasierten Ansätzen werden von Huhns und Singh (1998) und Wooldridge und Rao (1999) gesammelt. Texte über Multiagentensysteme bieten eine gute Einführung in viele Aspekte des Agentenentwurfs (Weiss, 2000a; Wooldridge, 2009). In den 1990er Jahren starteten mehrere Konferenzreihen, die sich mit Agenten beschäftigen, darunter der *International Workshop on Agent Theories, Architectures, and Languages* (ATAL), die *International Conference on Autonomous Agents* (AGENTS) und die *International Conference on Multi-Agent Systems* (ICMAS). Im Jahr 2002 fusionierten diese drei zu der *International Joint Conference on Autonomous Agents and Multi-Agent Systems* (AAMAS). Von 2000 bis 2012 gab es jährliche Workshops zum Thema *Agent-Oriented Software Engineering* (AOSE). Die Zeitschrift *Autonomous Agents and Multi-Agent Systems* wurde 1998 gegründet. Schließlich bietet *Dung Beetle Ecology* (Hanski und Cambefort, 1991) eine Fülle von interessanten Informationen über das Verhalten von Mistkäfern. Auf YouTube gibt es inspirierende Videoaufnahmen von ihren Aktivitäten.

Problemlösen durch Suchen

3

In diesem Kapitel erfahren wir, wie ein Agent mittels vorausschauender Entscheidungen eine Folge von Aktionen findet, mit der er sein Ziel erreichen kann.

Wenn die richtige Aktion nicht sofort ersichtlich ist, muss ein Agent möglicherweise *vorausplanen*: Er muss eine *Folge* von Aktionen in Betracht ziehen, die einen Pfad zu einem Zielzustand bilden. Ein solcher Agent wird als **problemlösender Agent** bezeichnet und der Rechenprozess, den er ausführt, heißt **Suche**.

Problemlösende Agenten verwenden **atomare** Repräsentationen, wie in Abschnitt 2.4.7 beschrieben – d. h., die Zustände der Welt werden als Ganzes betrachtet, ohne dass die interne Struktur für die problemlösenden Algorithmen sichtbar ist. Agenten, die **faktorisierte** oder **strukturierte** Repräsentationen von Zuständen verwenden, werden als **planende Agenten** bezeichnet, wir behandeln sie in den Kapiteln 7 und 11.

Wir werden verschiedene Suchalgorithmen behandeln. In diesem Kapitel betrachten wir nur die einfachsten Umgebungsarten: episodisch, einzelner Agent, vollständig beobachtbar, deterministisch, statisch, diskret und bekannt. Wir unterscheiden zwischen **informierten** Algorithmen, bei denen der Agent abschätzen kann, wie weit er vom Ziel entfernt ist, und **uninformierten** Algorithmen, bei denen keine solche Abschätzung verfügbar ist. In Kapitel 4 werden die Einschränkungen der Umgebungen gelockert und in Kapitel 5 werden mehrere Agenten betrachtet.

Dieses Kapitel verwendet die Konzepte der asymptotischen Komplexität (d. h. die $O(n)$-Notation). Leser, die mit diesen Konzepten nicht vertraut sind, sollten Anhang A zurate ziehen.

3.1 Problemlösende Agenten

Stellen Sie sich einen Agenten vor, der in seinem Urlaub durch Rumänien reist. Der Agent möchte sich die Sehenswürdigkeiten ansehen, sein Rumänisch verbessern, das Nachtleben genießen, einen Kater vermeiden und so weiter. Das Entscheidungsproblem ist also sehr komplex. Nehmen wir nun an, der Agent befindet sich gerade in der Stadt Arad und hat ein nicht erstattungsfähiges Flugticket ab Bukarest für den nächsten Tag. Der Agent studiert die Straßenschilder und stellt fest, dass es drei Straßen gibt, die aus Arad herausführen: eine in Richtung Sibiu, eine nach Timisoara und eine nach Zerind. Keiner dieser Orte ist das Ziel, d. h., falls der Agent nicht mit der Geografie Rumäniens vertraut ist, weiß er nicht, welcher Straße er folgen soll.[1]

Hat der Agent keine zusätzlichen Informationen – d. h., die Umgebung ist **unbekannt** –, so kann er nichts Besseres tun, als eine der Aktionen zufällig auszuführen. Diese unglückliche Situation wird in Kapitel 4 behandelt. In diesem Kapitel gehen wir davon aus, dass unsere Agenten immer Zugang zu Informationen über die Welt haben, wie z. B. die Karte in ► Abbildung 3.1. Mit diesen Informationen kann der Agent diesem Vier-Phasen-Problemlösungsprozess folgen:

- **Zielformulierung**: Der Agent wählt das **Ziel**, Bukarest zu erreichen. Ziele strukturieren das Verhalten, indem sie die Menge der Zielvorgaben und damit die zu berücksichtigenden Aktionen einschränken.
- **Problemformulierung**: Der Agent entwirft eine Beschreibung der Zustände und Aktionen, die notwendig sind, um das Ziel zu erreichen – ein abstraktes Modell des relevanten Teils

1 Wir gehen davon aus, dass es den meisten Lesern genauso geht und sie sich leicht in die Lage unseres ahnungslosen Agenten versetzen können. Wir entschuldigen uns bei den rumänischen Lesern, bei denen dieser didaktische Trick natürlich nicht greift.

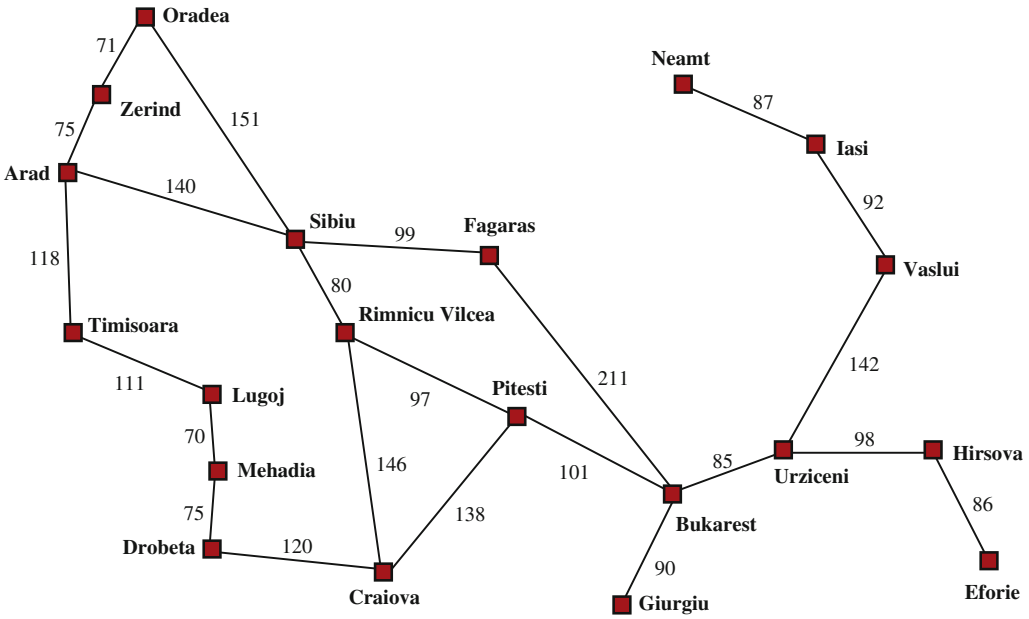


Abbildung 3.1: Eine vereinfachte Straßenkarte eines Teils von Rumänien (Entfernungen sind in Meilen angegeben).

der Welt. Für unseren Agenten ist ein gutes Modell, die Aktionen des Reisens von einer Stadt zu einer anderen Stadt zu betrachten, und daher ist die einzige Tatsache über den Zustand der Welt, die sich aufgrund einer Aktion ändert, die aktuelle Stadt.

- **Suche**: Bevor der Agent eine Aktion in der realen Welt ausführt, simuliert er Aktionsfolgen in seinem Modell und sucht so lange, bis er eine Folge findet, die das Ziel erreicht – diese wird als **Lösung** bezeichnet. Der Agent wird möglicherweise erst mehrere Folgen simulieren, die das Ziel nicht erreichen, aber schließlich wird er eine Lösung finden (z. B. von Arad nach Sibiu nach Fagaras nach Bukarest) oder er wird feststellen, dass keine Lösung möglich ist.
- **Ausführung**: Der Agent kann nun die Aktionen in der Lösung ausführen, eine nach der anderen.

Eine wichtige Eigenschaft in einer vollständig beobachtbaren, deterministischen, bekannten Umgebung ist, dass *die Lösung jedes Problems eine feste Abfolge von Aktionen ist:* Fahr nach Sibiu, dann nach Fagaras, dann nach Bukarest. Ist das Modell korrekt, so kann der Agent, sobald er eine Lösung gefunden hat, seine Perzepte ignorieren, während er die Aktionen ausführt – sozusagen die Augen schließen –, da die Lösung garantiert zum Ziel führt. Kontrolltheoretiker nennen dies ein **Open-Loop**-System: Das Ignorieren der Perzepte unterbricht die Schleife zwischen Agent und Umwelt. Besteht die Möglichkeit, dass das Modell falsch oder die Umgebung nichtdeterministisch ist, dann wäre der Agent sicherer, wenn er einen **Closed-Loop**-Ansatz verfolgen würde, der die Perzepte überwacht (siehe Abschnitt 4.4).

In teilweise beobachtbaren oder nichtdeterministischen Umgebungen wäre eine Lösung eine Verzweigungsstrategie, die verschiedene zukünftige Aktionen empfiehlt, je nachdem, welche Perzepte eintreffen. Zum Beispiel könnte der Agent planen, von Arad nach Sibiu zu fahren, aber er könnte einen Notfallplan brauchen, falls er zufällig in Zerind ankommt oder auf ein Schild mit der Aufschrift „Drum Închis“ (Straße gesperrt) trifft.

3.1.1 Suchprobleme und Lösungen

Ein **Suchproblem** kann formal durch folgende Elemente definiert werden:

- Eine Menge möglicher **Zustände**, in denen sich die Umgebung befinden kann. Wir nennen dies den **Zustandsraum**.
- Der **Anfangszustand**, in dem der Agent beginnt. Zum Beispiel: *Arad*.
- Eine Menge von einem oder mehreren **Zielzuständen**. Manchmal gibt es nur einen Zielzustand (z. B. *Bukarest*), manchmal gibt es eine kleine Menge alternativer Zielzustände und manchmal ist das Ziel durch eine Eigenschaft definiert, die für viele Zustände gilt (möglicherweise eine unendliche Anzahl). In einer Staubsaugerwelt könnte das Ziel beispielsweise sein, dass an keinem Ort Schmutz vorhanden ist, unabhängig von allen anderen Fakten über den Zustand. Wir können alle drei dieser Möglichkeiten berücksichtigen, indem wir eine IS-GOAL-Methode für ein Problem angeben. In diesem Kapitel sagen wir der Einfachheit halber manchmal „das Ziel", meinen damit aber „jeden der möglichen Zielzustände".
- Die **Aktionen**, die dem Agenten zur Verfügung stehen. Für einen gegebenen Zustand s gibt die Funktion ACTIONS(s) eine endliche[2] Menge von Aktionen zurück, die in s ausgeführt werden können. Wir sagen, dass jede dieser Aktionen in s **anwendbar** ist. Ein Beispiel:

$$\text{ACTIONS}(Arad) = \{NachSibiu, NachTimisoara, NachZerind\}$$

- Ein **Transitionsmodell**, das beschreibt, was jede Aktion bewirkt. RESULT(s, a) gibt den Zustand zurück, der aus der Ausführung von Aktion a im Zustand s resultiert. Zum Beispiel:

$$\text{RESULT}(Arad, NachZerind) = Zerind$$

- Eine **Aktionskostenfunktion**, ACTION-COST(s, a, s') (in Programmierschreibweise) oder $c(s, a, s')$ (in mathematischer Schreibweise), die die numerischen Kosten der Anwendung von Aktion a im Zustand s angibt, um Zustand s' zu erreichen. Ein problemlösender Agent sollte eine Kostenfunktion verwenden, die sein eigenes Leistungsmaß widerspiegelt; für Agenten, die eine Route finden, könnten die Kosten einer Aktion beispielsweise die Länge in Kilometern oder Meilen sein (wie in ► Abbildung 3.1 zu sehen) oder es könnte die Zeit sein, die benötigt wird, um die Aktion durchzuführen.

Eine Aktionsfolge bildet einen **Pfad**, und eine **Lösung** ist ein Pfad vom Anfangszustand zu einem Zielzustand. Wir nehmen an, dass die Aktionskosten additiv sind, d. h., die Gesamtkosten eines Pfads setzen sich aus den einzelnen Aktionskosten zusammen. Eine **optimale Lösung** hat die niedrigsten Pfadkosten unter allen Lösungen. In diesem Kapitel nehmen wir an, dass alle Aktionskosten positiv sind, um gewisse Komplikationen zu vermeiden.[3]

2 Für Probleme mit einer unendlichen Anzahl von Aktionen würden wir Techniken benötigen, die über den Inhalt dieses Kapitels hinausgehen.

3 Bei jedem Problem mit einem Zyklus mit negativen Nettokosten besteht die kostenoptimale Lösung darin, diesen Zyklus unendlich oft auszuführen. Die Bellman-Ford- und Floyd-Warshall-Algorithmen (werden hier nicht besprochen) behandeln Aktionen mit negativen Kosten, solange es keine negativen Zyklen gibt. Es ist einfach, Null-Kosten-Aktionen zu berücksichtigen, solange die Anzahl der aufeinanderfolgenden Null-Kosten-Aktionen begrenzt ist. Zum Beispiel könnten wir einen Roboter haben, bei dem zwar die Bewegungen etwas kosten, nicht jedoch die Drehung um 90°; die Algorithmen in diesem Kapitel können damit umgehen, solange nicht mehr als drei aufeinanderfolgende 90°-Drehungen erlaubt sind. Es gibt auch eine Komplikation bei Problemen, die eine unendliche Anzahl von beliebig kleinen Aktionskosten haben. Betrachten Sie eine Version des Zeno-Paradoxons, bei dem es eine Aktion gibt, mit der man sich auf halbem Weg zum Ziel bewegt, und zwar zu den halben Kosten der vorherigen Bewegung. Dieses Problem hat keine Lösung mit einer endlichen Anzahl von Aktionen, aber um zu verhindern, dass eine Suche eine unbegrenzte Anzahl von Aktionen ausführt, ohne das Ziel jemals ganz zu erreichen, können wir fordern, dass alle Aktionskosten mindestens ϵ sind, wobei ϵ ein kleiner positiver Wert ist.

Der Zustandsraum kann als **Graph** dargestellt werden, in dem die Knoten Zustände und die gerichteten Kanten zwischen ihnen Aktionen sind. Die in ► Abbildung 3.1 gezeigte Karte von Rumänien ist ein solcher Graph, wobei jede Straße zwei Aktionen bezeichnet, eine in jeder Richtung.

3.1.2 Probleme formulieren

Unsere Formulierung des Problems, nach Bukarest zu gelangen, ist ein **Modell** – eine abstrakte mathematische Beschreibung – und nichts Reales. Vergleichen Sie die einfache atomare Zustandsbeschreibung von *Arad* mit einer tatsächlichen Überlandfahrt, bei der der Zustand der Welt so viele Dinge umfasst: die Mitreisenden, das aktuelle Radioprogramm, die Landschaft vor dem Fenster, die Nähe von Ordnungshütern, die Entfernung zur nächsten Raststätte, der Zustand der Straße, das Wetter, der Verkehr und so weiter. All diese Überlegungen lassen wir in unserem Modell außen vor, weil sie irrelevant sind für das Problem, eine Route nach Bukarest zu finden.

Der Prozess, Details aus einer Darstellung zu entfernen, wird als **Abstraktion** bezeichnet. Eine gute Problemformulierung hat den richtigen Grad an Detailgenauigkeit. Wären die Aktionen auf der Ebene von „bewege den rechten Fuß einen Zentimeter nach vorne“ oder „drehe das Lenkrad um ein Grad nach links“, dann würde der Agent wahrscheinlich nie den Weg aus dem Parkplatz finden, geschweige denn nach Bukarest.

Können wir den angemessenen **Abstraktionsgrad** genauer bestimmen? Stellen Sie sich vor, dass die abstrakten Zustände und Aktionen, die wir gewählt haben, großen Mengen von detaillierten Weltzuständen und detaillierten Aktionsfolgen entsprechen. Betrachten wir nun eine Lösung des abstrakten Problems: zum Beispiel der Weg von Arad nach Sibiu nach Rimnicu Vilcea nach Pitesti nach Bukarest. Diese abstrakte Lösung entspricht einer großen Anzahl von ausführlicheren Pfaden. Zum Beispiel könnten wir auf der Fahrt von Sibiu nach Rimnicu Vilcea das Radio eingeschaltet haben und es dann für den Rest der Strecke ausschalten.

Die Abstraktion ist *gültig*, wenn wir jede abstrakte Lösung zu einer Lösung in der detaillierteren Welt ausarbeiten können; eine hinreichende Bedingung ist, dass es für jeden detaillierten Zustand, z. B. „in Arad“, einen detaillierten Pfad zu einem Zustand gibt, der beispielsweise „in Sibiu“ heißt, und so weiter.[4] Die Abstraktion ist *nützlich*, wenn die Ausführung jeder der Aktionen in der Lösung einfacher ist als das ursprüngliche Problem; in unserem Fall kann die Aktion „fahre von Arad nach Sibiu“ ohne weitere Suche oder Planung von einem durchschnittlichen Fahrer ausgeführt werden. Bei der Wahl einer guten Abstraktion geht es also darum, so viele Einzelheiten wie möglich zu entfernen, dabei aber die Validität zu erhalten und sicherzustellen, dass die abstrakten Aktionen leicht auszuführen sind. Ohne die Fähigkeit, sinnvolle Abstraktionen zu konstruieren, wären intelligente Agenten mit der realen Welt völlig überfordert.

3.2 Beispielprobleme

Der Problemlösungsansatz wurde auf einen umfangreichen Bereich von Aufgabenumgebungen angewendet. Wir listen hier einige der bekanntesten auf, wobei wir zwischen *standardisierten* und *realen* Problemen unterscheiden. Ein **standardisiertes Problem** dient der Veranschaulichung oder Übung verschiedener Problemlösungsmethoden. Es kann knapp und genau beschrieben werden und eignet sich daher als Maßstab für Forscher, um die Leistung von Algorithmen zu vergleichen. Ein **reales Problem**, wie z. B. die Roboternavigation, ist ein Problem, dessen Lösungen tatsächlich verwendet werden und dessen Formulierung extrem spezifisch und nicht standardisiert ist, da beispielsweise jeder Roboter unterschiedliche Sensoren hat, die unterschiedliche Daten liefern.

4 Siehe Abschnitt 11.4.

3.2.1 Standardisierte Probleme

Ein **Gitterwelt**-Problem ist eine zweidimensionale rechteckige Anordnung von quadratischen Zellen, in denen sich Agenten von Zelle zu Zelle bewegen können. Typischerweise kann sich der Agent zu jeder hindernisfreien benachbarten Zelle bewegen – horizontal oder vertikal und in einigen Problemen auch diagonal. Zellen können Objekte enthalten, die der Agent aufnehmen, verschieben oder auf die er anderweitig einwirken kann; eine Wand oder ein anderes unüberwindbares Hindernis in einer Zelle hindern einen Agenten daran, sich in diese Zelle zu begeben. Die **Staubsaugerwelt** aus Abschnitt 2.1 kann als Gitterwelt-Problem wie folgt formuliert werden:

- **Zustände**: Ein Zustand dieser Welt gibt Auskunft darüber, welche Objekte sich in welchen Zellen befinden. Für die Staubsaugerwelt sind die Objekte der Agent und eventueller Schmutz. In der einfachen Zwei-Zellen-Version kann sich der Agent in einer der beiden Zellen befinden und jede Zelle kann entweder Schmutz enthalten oder nicht, es gibt also $2 \cdot 2 \cdot 2 = 8$ Zustände (▶ Abbildung 3.2). Im Allgemeinen hat eine Staubsaugerumgebung mit n Zellen $n \cdot 2^n$ Zustände.

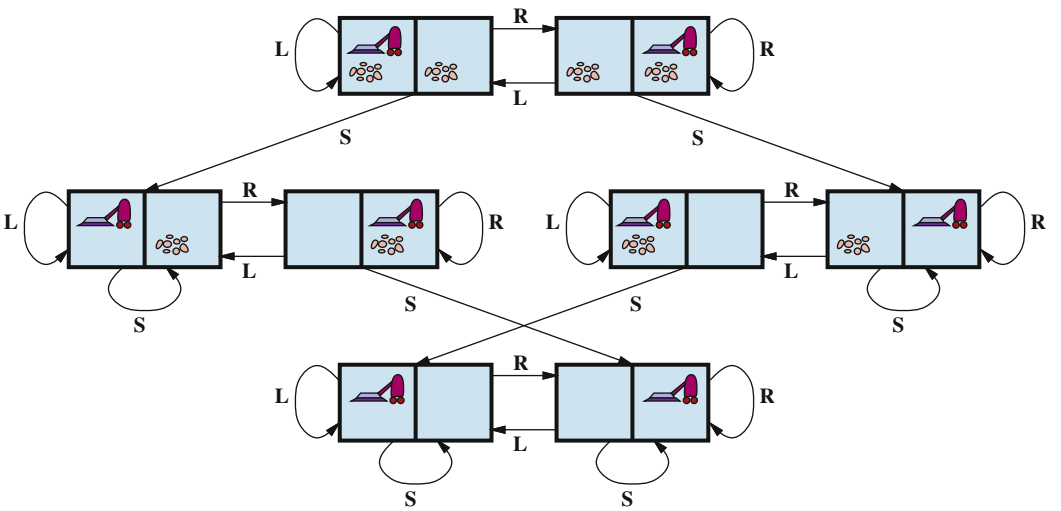


Abbildung 3.2: Der Graph des Zustandsraums für die zweizellige Staubsaugerwelt. Es gibt 8 Zustände und drei Aktionen für jeden Zustand: L = *Links*, R = *Rechts*, S = *Saugen*.

- **Anfangszustand**: Als Anfangszustand kann ein beliebiger Zustand festgelegt werden.
- **Aktionen**: In der Zwei-Zellen-Welt haben wir drei Aktionen definiert: *Saugen*, *Links*-Bewegung und *Rechts*-Bewegung. In einer zweidimensionalen mehrzelligen Welt benötigen wir mehr Bewegungsaktionen. Wir könnten *Aufwärts* und *Abwärts* hinzufügen, wodurch wir vier **absolute** Bewegungsaktionen hätten, oder wir könnten zu **egozentrischen Aktionen** übergehen, die relativ zum Standpunkt des Agenten definiert sind – zum Beispiel *Vorwärts*, *Rückwärts*, *DrehenRechts* und *DrehenLinks*.
- **Transitionsmodell**: *Saugen* entfernt jeglichen Schmutz aus der Zelle des Agenten; *Vorwärts* bewegt den Agenten um eine Zelle in die Richtung, in die er blickt, es sei denn, er trifft auf eine Wand, in diesem Fall hat die Aktion keine Wirkung. *Rückwärts* bewegt den Agenten in die entgegengesetzte Richtung, während *DrehenRechts* und *DrehenLinks* seine Blickrichtung um 90° ändern.

- **Zielzustände**: Die Zustände, in denen jede Zelle sauber ist.
- **Aktionskosten**: Jede Aktion kostet 1.

Eine weitere Art der Gitterwelt ist das **Sokoban-Puzzle**, bei dem das Ziel des Agenten darin besteht, eine Anzahl von Kisten, die über das Gitter verstreut sind, an bestimmte Lagerorte zu schieben. Es kann maximal eine Kiste pro Zelle geben. Wenn sich ein Agent in eine Zelle mit einer Kiste vorwärts bewegt und sich auf der anderen Seite der Kiste eine leere Zelle befindet, dann bewegen sich sowohl die Kiste als auch der Agent vorwärts. Der Agent kann eine Kiste nicht in eine andere Kiste oder eine Wand schieben. Für eine Welt mit n Zellen ohne Hindernisse und b Kisten gibt es $n \times n!/(b!(n-b)!)$ Zustände; zum Beispiel gibt es auf einem 8×8-Gitter mit einem Dutzend Kisten über 200 Billionen Zustände.

Bei einem **Schiebepuzzle** wird eine Reihe von Spielsteinen (manchmal auch Blöcke oder Teile genannt) in einem Gitter mit einem oder mehreren Leerräumen angeordnet, sodass einige der Spielsteine in den Leerraum gleiten können. Eine Variante ist das Puzzle „Rush Hour“, bei dem Autos und Lkws auf einem 6×6-Gitter verschoben werden, um ein Auto aus dem Stau zu befreien. Die wohl bekannteste Variante ist das **8-Puzzle** (► Abbildung 3.3), das aus einem 3×3-Gitter mit acht nummerierten Spielsteinen und einem leeren Feld besteht, sowie das **15-Puzzle** auf einem 4×4-Gitter. Das Ziel ist es, einen bestimmten Zielzustand zu erreichen, wie beispielsweise den Zustand rechts in der Abbildung. Die Standardformulierung des 8-Puzzles lautet:

- **Zustände**: Eine Zustandsbeschreibung spezifiziert die Lage der einzelnen Spielsteine.
- **Anfangszustand**: Jeder beliebige Zustand kann als Anfangszustand festgelegt werden. Beachten Sie, dass eine Paritätseigenschaft den Zustandsraum unterteilt – jedes gegebene Ziel kann von genau der Hälfte der möglichen Anfangszustände erreicht werden (siehe Übung 3.6 auf der Website).
- **Aktionen**: In der physischen Welt handelt es sich zwar um einen Spielstein, der verschoben wird, doch die einfachste Art, eine Aktion zu beschreiben, ist sich vorzustellen, dass das leere Feld eine *Links*-, *Rechts*-, *Oben*- oder *Unten*-Bewegung ausführt. Wenn das Leerfeld am Rand oder in einer Ecke liegt, sind nicht alle Aktionen anwendbar.
- **Transitionsmodell**: Bildet einen Zustand und eine Aktion auf einen resultierenden Zustand ab; wenn wir z. B. *Links* auf den Startzustand in ► Abbildung 3.3 anwenden, so sind beim resultierenden Zustand die 5 und das Leerfeld vertauscht.
- **Zielzustand**: Obwohl jeder beliebige Zustand das Ziel sein könnte, nehmen wir normalerweise einen Zustand, bei dem die Zahlen in der richtigen Reihenfolge sind wie in ► Abbildung 3.3.
- **Aktionskosten**: Jede Aktion kostet 1.

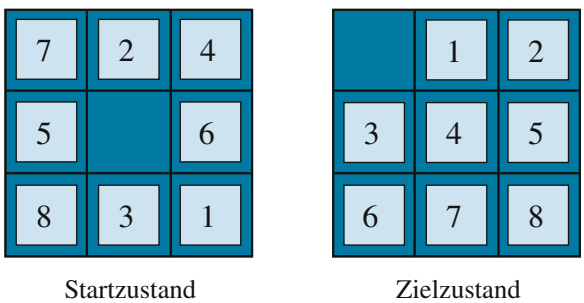


Abbildung 3.3: Ein typisches Beispiel für ein 8-Puzzle.

Beachten Sie, dass jede Problemformulierung Abstraktionen beinhaltet. Die 8-Puzzle-Aktionen werden auf ihre Anfangs- und Endzustände abstrahiert, wobei die Zwischenpositionen, bei denen der Spielstein verschoben wird, ignoriert werden. Wir haben Aktionen wie das Schütteln des Bretts, wenn die Spielsteine sich verklemmt haben, „wegabstrahiert“ und das Ausbauen der Steine mit einem Messer mit anschließendem Wiedereinsetzen ausgeschlossen. Übrig bleibt eine Beschreibung der Regeln, die alle Details der physikalischen Manipulationen vermeidet.

Unser letztes standardisiertes Problem wurde von Donald Knuth (1964) entwickelt und veranschaulicht, wie unendliche Zustandsräume entstehen können. Knuth vermutete, dass man von der Zahl 4 aus durch eine Folge von Quadratwurzel-, Fakultät- und Abrundungsoperationen (Gaußklammer) jede beliebige positive ganze Zahl erreichen kann. Zum Beispiel kann man folgendermaßen von der 4 zur 5 kommen:

$$\left\lfloor \sqrt{\sqrt{\sqrt{\sqrt{\sqrt{(4!)!}}}}} \right\rfloor = 5$$

Die Problemstellung ist einfach:

- **Zustände**: positive reelle Zahlen
- **Anfangszustand**: 4
- **Aktionen**: Anwenden von Quadratwurzel-, Fakultät- oder Abrundungsoperationen (Fakultät nur für ganze Zahlen)
- **Transitionsmodell**: wie durch die mathematischen Definitionen der Operationen gegeben
- **Zielzustand**: die gewünschte positive ganze Zahl
- **Aktionskosten**: Jede Aktion kostet 1.

Der Zustandsraum für dieses Problem ist unendlich: Für jede ganze Zahl größer als 2 wird der Fakultätsoperator immer eine größere ganze Zahl liefern. Das Problem ist interessant, weil es sehr große Zahlen untersucht: Der kürzeste Weg zu 5 führt über (4!)! = 620.448.401.733.239.439.360.000. Unendliche Zustandsräume treten häufig bei Aufgaben auf, die mathematische Ausdrücke, Schaltkreise, Beweise, Programme und andere rekursiv definierte Objekte generieren.

3.2.2 Probleme der realen Welt

Wir haben bereits gesehen, dass das Problem, eine Route zu finden (**Wegfindungsproblem**), durch die vorgegebenen Orte und die Übergänge entlang der Kanten zwischen ihnen definiert ist. Algorithmen zur Wegfindung werden in vielen Anwendungen eingesetzt. Einige, wie z. B. Websites und im Auto eingebaute Systeme, die Richtungsanweisungen geben, sind relativ einfache Erweiterungen des Rumänien-Beispiels. (Die Hauptkomplikationen sind unterschiedliche Kosten aufgrund von verkehrsbedingten Verzögerungen sowie Umleitungen aufgrund von Straßensperrungen.) Andere, wie beispielsweise das Routing von Videostreams in Computernetzen, militärische Operationsplanung und die Reiseplanung von Fluggesellschaften, beinhalten erheblich komplexere Spezifikationen. Betrachten Sie die Probleme bei Flugreisen, die von einer Website zur Reiseplanung gelöst werden müssen:

- **Zustände**: Jeder Zustand umfasst natürlich eine Position (z. B. einen Flughafen) und die aktuelle Zeit. Da die Kosten einer Aktion (eines Flugabschnitts) von früheren Abschnitten, ihren Flugtarifen und ihrem Status als nationaler oder internationaler Flug abhängen können, muss der Zustand außerdem zusätzliche Informationen über diese „historischen“ Aspekte erfassen.

- **Anfangszustand**: Der Heimatflughafen des Benutzers.
- **Aktionen**: Einen beliebigen Flug vom aktuellen Standort in einer beliebigen Sitzklasse nehmen, der später als die aktuelle Uhrzeit startet und genügend Zeit für einen Transfer innerhalb des Flughafens lässt, falls erforderlich.
- **Transitionsmodell**: Der Zustand, der sich nach Ausführen eines Flugs ergibt, hat das Flugziel als neue Position und die Ankunftszeit des Flugs als neue Zeit.
- **Zielzustand**: Eine Stadt als Bestimmungsort. Manchmal kann das Ziel auch komplexer sein, z. B. „mit einem Nonstop-Flug am Zielort ankommen".
- **Aktionskosten**: Eine Kombination aus monetären Kosten, Wartezeit, Flugzeit, Zoll- und Einwanderungsprozeduren, Sitzqualität, Tageszeit, Flugzeugtyp, Bonuspunkten für Vielflieger und so weiter.

Kommerzielle Reiseberatungssysteme verwenden eine Problemformulierung dieser Art – mit vielen zusätzlichen Komplikationen, die entstehen, um die verworrenen Tarifstrukturen der Fluggesellschaften zu handhaben. Jeder erfahrene Reisende weiß jedoch, dass nicht alle Flugreisen nach Plan verlaufen. Ein wirklich gutes System sollte Notfallpläne enthalten – was passiert, wenn dieser Flug Verspätung hat und der Anschluss verpasst wird?

Tourenplanungsprobleme beschreiben nicht ein einzelnes Ziel, sondern eine Reihe von Orten, die besucht werden müssen. Das **Problem des Handlungsreisenden** (*Traveling Salesperson Problem*, TSP) ist ein Tourenplanungproblem, bei dem jede Stadt auf einer Karte besucht werden muss. Das Ziel ist es, eine Tour mit Kosten $< C$ zu finden (oder in der Optimierungsversion, eine Tour mit den geringstmöglichen Kosten zu finden). Es wurden enorme Anstrengungen unternommen, um die Fähigkeiten der TSP-Algorithmen zu verbessern. Die Algorithmen können auch auf Fahrzeugflotten erweitert werden. Ein Such- und Optimierungsalgorithmus für die Routenplanung von Schulbussen in Boston sparte beispielsweise 5 Millionen US-Dollar, reduzierte das Verkehrsaufkommen und die Luftverschmutzung und sparte sowohl Fahrern als auch den Schülern Zeit (Bertsimas *et al.*, 2019). Neben der Planung von Fahrten wurden Suchalgorithmen auch für Aufgaben wie die Planung der Bewegungen von automatischen Leiterplattenbohrern und von Maschinen zur Regalbestückung in Werkshallen eingesetzt.

Ein **VLSI-Layoutproblem** erfordert die Positionierung von Millionen von Komponenten und Verbindungen auf einem Chip, um Fläche, Schaltverzögerungen und Streukapazitäten zu minimieren und die Produktionsausbeute zu maximieren. Das Layout-Problem kommt nach der logischen Entwurfsphase und wird normalerweise in zwei Teile aufgeteilt: **Zellen-Layout** und **Kanal-Routing**. Beim Zellen-Layout werden die elementaren Komponenten der Schaltung in Zellen gruppiert, von denen jede eine bestimmte Funktion ausführt. Jede Zelle hat eine feste Gehäusegeometrie (Größe und Umriss) und benötigt eine bestimmte Anzahl von Verbindungen zu den anderen Zellen. Ziel ist es, die Zellen so auf dem Chip zu platzieren, dass sie sich nicht überlappen und dass zwischen den Zellen Platz für die Verbindungsdrähte ist. Beim Kanal-Routing wird für jeden Draht ein bestimmter Weg durch die Lücken zwischen den Zellen gefunden. Diese Suchprobleme sind extrem komplex, doch es ist absolut sinnvoll, sie zu lösen.

Die **Roboternavigation** ist eine Verallgemeinerung des zuvor beschriebenen Problems der Wegfindung. Ein Roboter muss nicht bestimmten Pfaden folgen (wie z. B. den Straßen in Rumänien), sondern kann umherwandern und so seine eigenen Pfade erstellen. Für einen kreisförmigen Roboter, der sich auf einer ebenen Fläche bewegt, ist der Raum im Wesentlichen zweidimensional. Hat der Roboter Arme und Beine, die ebenfalls gesteuert werden müssen, dann wird der Suchraum mehrdimensional – eine Dimension für jeden Gelenkwinkel. Es sind fortschrittliche Techniken erforderlich, um den im Wesentlichen stetigen Suchraum endlich zu machen (siehe Kapitel 26). Zusätzlich zu der Komplexität des Problems müssen reale

Roboter auch auf Fehler in ihren Sensormessungen und den Motorsteuerungen, mit teilweiser Beobachtbarkeit und mit anderen Agenten, die die Umgebung verändern könnten, reagieren.

Die **automatische Montageablaufsteuerung** komplexer Objekte (wie z. B. Elektromotoren) durch einen Roboter ist seit den 1970er Jahren gängige Branchenpraxis. Algorithmen finden zunächst eine durchführbare Montagereihenfolge und arbeiten dann an der Optimierung des Prozesses. Die Minimierung des Anteils an manueller menschlicher Arbeit am Fließband kann zu erheblichen Zeit- und Kosteneinsparungen führen. Bei Montageablaufproblemen besteht das Ziel darin, eine Reihenfolge zu finden, in der die Teile eines Objekts zusammengebaut werden. Wenn die falsche Reihenfolge gewählt wird, gibt es keine Möglichkeit, ein Teil später im Ablauf hinzuzufügen, ohne bereits geleistete Arbeit rückgängig zu machen. Das Überprüfen einer Aktion in der Abfolge auf Durchführbarkeit ist ein schwieriges geometrisches Suchproblem, das eng mit der Roboternavigation verwandt ist. Daher ist die Generierung von zulässigen Aktionen der teure Teil der Montageablaufsteuerung. Jeder praktische Algorithmus darf nicht den gesamten, sondern nur einen winzigen Teil des Zustandsraums untersuchen. Ein wichtiges Montageproblem ist der **Proteinentwurf**, bei dem das Ziel darin besteht, eine Sequenz von Aminosäuren zu finden, die sich zu einem dreidimensionalen Protein mit den passenden Eigenschaften faltet, um eine Krankheit zu heilen.

3.3 Suchalgorithmen

Ein **Suchalgorithmus** bekommt ein Suchproblem als Eingabe und gibt entweder eine Lösung oder eine Fehlermeldung zurück. In diesem Kapitel betrachten wir Algorithmen, die einen **Suchbaum** über den Zustandsraumgraphen legen, verschiedene Pfade vom Anfangszustand aus bilden und versuchen, einen Pfad zu finden, der einen Zielzustand erreicht. Jeder **Knoten** im Suchbaum entspricht einem Zustand im Zustandsraum und die Kanten im Suchbaum entsprechen Aktionen. Die Wurzel des Baums stellt den Anfangszustand des Problems dar.

Es ist wichtig, den Unterschied zwischen dem Zustandsraum und dem Suchbaum zu verstehen. Der Zustandsraum beschreibt die (möglicherweise unendliche) Menge der Zustände in der Welt sowie die Aktionen, die Übergänge von einem Zustand zu einem anderen ermöglichen. Der Suchbaum beschreibt Pfade zwischen diesen Zuständen, die zum Ziel führen. Der Suchbaum kann mehrere Pfade zu jedem (und damit mehrere Knoten für jeden) beliebigen Zustand haben, aber jeder Knoten im Baum hat einen eindeutigen Pfad zurück zur Wurzel (wie in allen Bäumen).

► Abbildung 3.4 zeigt die ersten Schritte bei der Suche nach einem Pfad von Arad nach Bukarest. Der Wurzelknoten des Suchbaums befindet sich im Anfangszustand, *Arad*. Wir können den Knoten **expandieren**, indem wir die verfügbaren AKTIONEN für diesen Zustand betrachten und die RESULT-Funktion verwenden, um zu sehen, wohin diese Aktionen führen. Und wir können für jeden der resultierenden Zustände einen neuen Knoten **erzeugen** (**Kindknoten** oder **Nachfolgerknoten** genannt). Jeder Kindknoten hat *Arad* als seinen **Elternknoten**.

Nun müssen wir auswählen, welchen dieser drei Kindknoten wir als Nächstes untersuchen. Dies ist das Wesen der Suche – erst einmal eine Option zu verfolgen und sich die anderen für später vorzubehalten. Angenommen wir wählen, zuerst Sibiu zu expandieren. ► Abbildung 3.4 (unten) zeigt das Ergebnis: eine Menge von 6 nicht expandierten Knoten (grün und fett umrandet). Wir nennen dies die **Grenze** des Suchbaums. Wir sagen, dass jeder Zustand, für den ein Knoten erzeugt wurde, **erreicht** (*reached*) wurde (unabhängig davon, ob dieser Knoten expandiert wurde oder nicht).[5] ► Abbildung 3.5 zeigt den Suchbaum, der auf den Zustandsraumgraphen gelegt wurde.

5 Einige Autoren bezeichnen die Grenze als **offene Liste**, was sowohl geografisch weniger aussagekräftig als auch rechnerisch weniger geeignet ist, da eine Warteschlange hier effizienter ist als eine Liste. Diese Autoren verwenden den Begriff **geschlossene Liste**, um sich auf die Menge der zuvor expandierten Knoten zu beziehen, was in unserer Terminologie den *erreichten* Knoten abzüglich der *Grenze* entspricht.

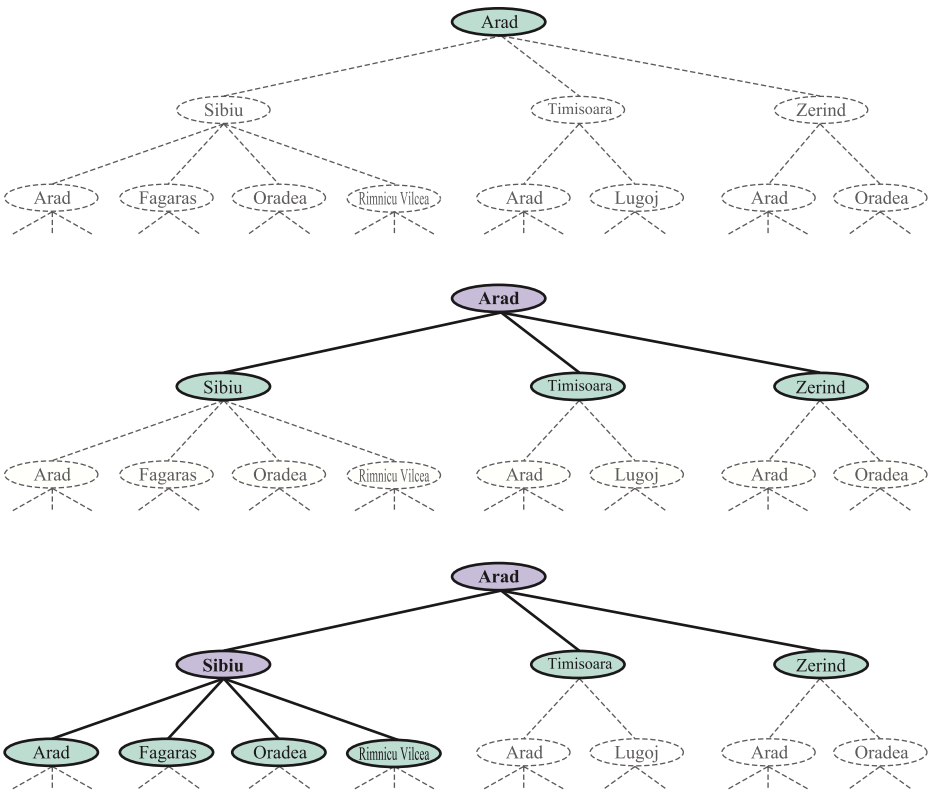


Abbildung 3.4: Drei partielle Suchbäume, um eine Route von Arad nach Bukarest zu finden. Knoten, die *expandiert* wurden, sind violett mit fett gedruckten Namen; Knoten auf der Grenze, die *erzeugt*, aber noch nicht expandiert wurden, sind grün; die Menge der Zustände, die diesen beiden Knotentypen entsprechen, gilt als *erreicht*. Knoten, die als Nächstes generiert werden könnten, sind mit hell gestrichelten Linien dargestellt. Beachten Sie, dass es im unteren Baum einen Zyklus von Arad nach Sibiu nach Arad gibt; das kann kein optimaler Pfad sein, also sollte die Suche von dort aus nicht fortgesetzt werden.

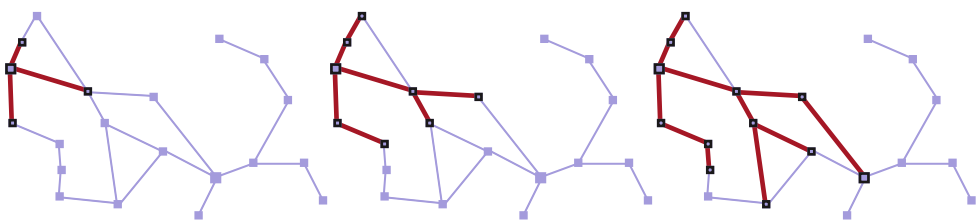


Abbildung 3.5: Eine Folge von Suchbäumen, die durch eine Graphensuche für das Rumänien-Problem aus Abbildung 3.1 generiert wurde. In jeder Stufe haben wir jeden Knoten auf der Grenze expandiert, indem wir jeden Pfad mit allen anwendbaren Aktionen erweitert haben, die nicht zu einem Zustand führen, der bereits erreicht wurde. Beachten Sie, dass in der dritten Stufe die oberste Stadt (Oradea) zwei Nachfolger hat, die beide bereits durch andere Pfade erreicht wurden, sodass keine Pfade von Oradea aus erweitert werden.

Beachten Sie, dass die Grenze zwei Regionen des Zustandsraumgraphen **trennt**: eine innere Region, in der jeder Zustand expandiert wurde, und eine äußere Region mit Zuständen, die noch nicht erreicht wurden. Diese Eigenschaft ist in ► Abbildung 3.6 dargestellt.

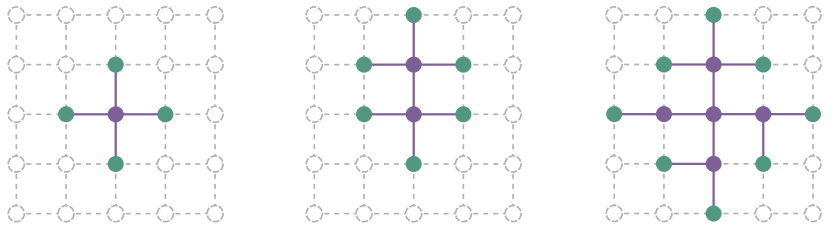


Abbildung 3.6: Die Trennungseigenschaft der Graphensuche, dargestellt an einem Problem mit rechteckigem Gitter. Die Grenze (grün) trennt das Innere (violett) vom Äußeren (schwach gestrichelt). Die Grenze ist die Menge der Knoten (und der entsprechenden Zustände), die erreicht, aber noch nicht expandiert wurden; das Innere ist die Menge der Knoten (und der entsprechenden Zustände), die expandiert wurden; und das Äußere ist die Menge der Zustände, die noch nicht erreicht wurden. In (a) wurde nur die Wurzel expandiert. In (b) ist der oberste Grenzknoten expandiert. In (c) wurden die übrigen Nachfolger der Wurzel im Uhrzeigersinn expandiert.

3.3.1 Bestensuche

Wie wird entschieden, welcher Knoten der Grenze als Nächstes erweitert werden soll? Ein sehr allgemeiner Ansatz wird als **Bestensuche** (*Best-First Search*, BFS) bezeichnet, bei der wir einen Knoten n mit dem minimalen Wert einer **Evaluierungsfunktion** $f(n)$ auswählen. ► Abbildung 3.7 zeigt den Algorithmus. Bei jeder Iteration wählen wir einen Knoten auf der Grenze mit minimalem $f(n)$-Wert, geben ihn zurück, wenn sein Zustand ein Zielzustand ist, und wenden andernfalls EXPAND an, um Kindknoten zu erzeugen. Jeder Kindknoten wird der Grenze hinzugefügt, falls er vorher nicht erreicht wurde, oder er wird erneut hinzugefügt, wenn er jetzt mit einem Pfad erreicht wird, der geringere Pfadkosten als jeder vorherige Pfad hat. Der Algorithmus gibt entweder eine Fehlermeldung oder einen Knoten zurück, der einen Pfad zu einem Ziel darstellt. Durch die Verwendung verschiedener $f(n)$-Funktionen erhalten wir unterschiedliche Algorithmen, die in diesem Kapitel behandelt werden.

```
function BEST-FIRST-SEARCH(problem, f) returns einen Knoten als Lösung oder failure
  node ← NODE(STATE=problem.INITIAL)
  frontier ← eine Prioritätswarteschlange, sortiert durch f, mit node als ein Element
  reached ← eine Nachschlagtabelle, mit einem Eintrag für den Schlüssel problem.INITIAL
                                                                  und Wert node
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if s ist nicht in reached or child.PATH-COST < reached[s].PATH-COST then
        reached[s] ← child
        füge child zu frontier hinzu
  return failure

function EXPAND(problem, node) yields nodes
  s ← node.STATE
  for each action in problem.ACTIONS(s) do
    s' ← problem.RESULT(s, action)
    cost ← node.PATH-COST + problem.ACTION-COST(s, action, s')
    yield NODE(STATE=s', PARENT=node, ACTION=action, PATH-COST=cost)
```

Abbildung 3.7: Der Algorithmus zur Bestensuche und die Funktion zur Expansion eines Knotens. Die hier verwendeten Datenstrukturen sind in Abschnitt 3.3.2 beschrieben. Für **yield** siehe Anhang B.

3.3.2 Datenstrukturen für die Suche

Suchalgorithmen benötigen eine Datenstruktur, um den Überblick über den Suchbaum zu behalten. Ein **Knoten** im Baum wird durch eine Datenstruktur mit vier Komponenten dargestellt:

- *node*.STATE: der Zustand, dem der Knoten entspricht;
- *node*.PARENT: der Knoten im Baum, der diesen Knoten erzeugt hat;
- *node*.ACTION: die Aktion, die auf den Elternzustand angewendet wurde, um diesen Knoten zu erzeugen;
- *node*.PATH-COST: die Gesamtkosten des Pfads vom Anfangszustand zu diesem Knoten. In mathematischen Formeln verwenden wir $g(node)$ als Synonym für PATH-COST.

Wenn wir die PARENT-Zeiger von einem Knoten zurückverfolgen, können wir die Zustände und Aktionen entlang des Pfads zu diesem Knoten wiederherstellen. Wenn Sie dies von einem Zielknoten aus tun, erhalten Sie die Lösung.

Wir brauchen eine Datenstruktur, um die **Grenze** zu speichern. Die geeignete Wahl ist eine Art **Warteschlange**, denn die Operationen auf einer Grenze sind:

- IS-EMPTY(*frontier*) – gibt *true* (wahr) genau dann zurück, wenn es keine Knoten in der Grenze gibt.
- POP(*frontier*) – entfernt den obersten Knoten von der Grenze und gibt ihn zurück.
- TOP(*frontier*) – gibt den obersten Knoten der Grenze zurück (entfernt ihn aber nicht).
- ADD(*node*, *frontier*) – fügt den Knoten an seinem richtigen Platz in der Warteschlange ein.

In Suchalgorithmen werden drei Arten von Warteschlangen verwendet:

- In einer **Prioritätswarteschlange** wird die Funktion POP zuerst auf den Knoten mit den geringsten Kosten gemäß einer Evaluierungsfunktion f angewandt. Sie wird bei der Bestensuche verwendet.
- In einer **FIFO-Warteschlange** oder First-In-First-Out-Warteschlange wird POP zuerst auf den Knoten angewandt, der als Erster in die Warteschlange aufgenommen wurde; wir werden sehen, dass dies bei der Breitensuche verwendet wird.
- Eine **LIFO-Warteschlange** oder Last-In-First-Out-Warteschlange (auch als **Stapel** oder **Stack** bezeichnet) entfernt zuerst den zuletzt hinzugefügten Knoten; wir werden sehen, dass sie bei der Tiefensuche eingesetzt wird.

Die erreichten Zustände können als Nachschlagetabelle (z. B. als Hash-Tabelle) gespeichert werden, wobei jeder Schlüssel ein Zustand und jeder Wert der Knoten für diesen Zustand ist.

3.3.3 Redundante Pfade

Der in ► Abbildung 3.4 (unten) dargestellte Suchbaum enthält einen Pfad von Arad nach Sibiu und wieder zurück nach Arad. Wir sagen, dass *Arad* ein **wiederholter Zustand** im Suchbaum ist, der in diesem Fall durch einen **Zyklus** (auch **schleifenförmiger Pfad** genannt) erzeugt wird. So hat der Zustandsraum zwar nur 20 Zustände, doch der vollständige Suchbaum ist *unendlich*, weil es keine Begrenzung gibt, wie oft man eine Schleife durchlaufen kann.

Ein Zyklus ist ein Spezialfall eines **redundanten Pfads**. Zum Beispiel können wir nach Sibiu über den Pfad Arad–Sibiu (140 Meilen) oder den Pfad Arad–Zerind–Oradea–Sibiu (297 Meilen) kommen. Dieser zweite Pfad ist redundant – er ist nur eine schlechtere Möglichkeit, um zum gleichen Zustand zu gelangen – und muss bei unserer Suche nach optimalen Pfaden nicht berücksichtigt werden.

Betrachten Sie einen Agenten in einer 10×10-Gitterwelt, der sich auf jedes von 8 benachbarten Feldern bewegen kann. Wenn es keine Hindernisse gibt, kann der Agent jedes der 100 Quadrate in 9 Zügen oder weniger erreichen. Doch die Anzahl der Pfade mit Länge 9 ist fast 8^9 (wegen der Kanten des Gitters ein bisschen weniger) bzw. mehr als 100 Millionen. Mit anderen Worten, im Durchschnitt kann eine Zelle über mehr als eine Million redundanter Pfade der Länge 9 erreicht werden und wenn wir redundante Pfade eliminieren, können wir eine Suche etwa eine Million Mal schneller beenden. Wie heißt es so schön: *Algorithmen, die sich nicht an die Vergangenheit erinnern können, sind dazu verdammt, sie zu wiederholen.* Es gibt drei Wege, dieses Problem anzugehen.

Erstens können wir uns alle zuvor erreichten Zustände merken (wie bei der Bestensuche), was uns erlaubt, alle redundanten Pfade zu erkennen und nur den besten Pfad zu jedem Zustand zu behalten. Dies ist für Zustandsräume geeignet, in denen es viele redundante Pfade gibt, und ist die bevorzugte Wahl, wenn die Tabelle der erreichten Zustände in den Speicher passt.

Zweitens: Wir müssen uns nicht darum sorgen, die Vergangenheit zu wiederholen. Es gibt einige Problemstellungen, bei denen es selten oder unmöglich ist, dass zwei Pfade denselben Zustand erreichen. Ein Beispiel wäre ein Montageproblem, bei dem durch jede Aktion ein Teil zu einer sich entwickelnden Baugrupppe hinzufügt wird und bei dem es eine Reihenfolge der Teile gibt, sodass es möglich ist, erst *A* und dann *B* hinzuzufügen, aber nicht erst *B* und dann *A*. Bei solchen Problemen können wir Speicherplatz sparen, wenn wir erreichte Zustände *nicht* verfolgen und *nicht* auf redundante Pfade prüfen. Wir nennen einen Suchalgorithmus eine **Graphensuche**, wenn er auf redundante Pfade prüft, und eine **baumartige Suche**[6], wenn er nicht prüft. Der Algorithmus BEST-FIRST-SEARCH in ► Abbildung 3.7 ist ein Algorithmus zur Graphensuche; wenn wir alle Verweise auf *reached* entfernen, erhalten wir eine baumartige Suche, die weniger Speicher benötigt, aber redundante Pfade zum gleichen Zustand untersucht und daher langsamer läuft.

Drittens können wir einen Kompromiss eingehen und zwar auf Zyklen, aber nicht auf redundante Pfade im Allgemeinen prüfen. Da jeder Knoten eine Kette von Elternzeigern hat, können wir ohne zusätzlichen Speicherbedarf auf Zyklen prüfen, indem wir die Kette der Eltern nach oben verfolgen, um festzustellen, ob der Zustand am Ende des Pfads früher im Pfad schon einmal aufgetaucht ist. Einige Implementierungen folgen dieser Kette den ganzen Weg nach oben und eliminieren so alle Zyklen; andere Implementierungen folgen nur ein paar Gliedern (z. B. zum Elternteil, zu Großeltern und Urgroßeltern) und benötigen so nur eine konstante Zeit, um alle kurzen Zyklen zu eliminieren (und verlassen sich auf andere Mechanismen, um lange Zyklen zu behandeln).

3.3.4 Leistungsmessung der Problemlösung

Bevor wir uns mit dem Entwurf verschiedener Suchalgorithmen befassen, betrachten wir die Kriterien, die verwendet werden, um einen Algorithmus auszuwählen. Wir können die Leistung eines Algorithmus auf vier Arten bewerten:

- **Vollständigkeit**: Findet der Algorithmus garantiert eine Lösung, falls es eine gibt, und gibt er korrekt eine Fehlermeldung zurück, falls es keine gibt?
- **Kostenoptimalität**: Findet er eine Lösung mit den geringsten Pfadkosten aller Lösungen?[7]

6 Wir sagen „baumartige Suche“, weil der Zustandsraum immer noch derselbe Graph ist, egal wie wir ihn durchsuchen; wir entscheiden uns nur dafür, ihn *wie* einen Baum zu behandeln, also mit nur einem Pfad von jedem Knoten zurück zur Wurzel.

7 Einige Autoren verwenden den Begriff „Zulässigkeit“ für die Eigenschaft, die kostengünstigste Lösung zu finden, und einige benutzen einfach „Optimalität“, doch dies kann mit anderen Arten von Optimalität verwechselt werden.

- **Zeitkomplexität**: Wie lange dauert es, eine Lösung zu finden? Dies kann in Sekunden gemessen werden oder abstrakter durch die Anzahl der betrachteten Zustände und Aktionen.
- **Speicherplatzkomplexität**: Wie viel Speicher wird für die Durchführung der Suche benötigt?

Um das Konzept der Vollständigkeit zu verstehen, betrachten Sie ein Suchproblem mit einem einzigen Ziel. Dieses Ziel könnte überall im Zustandsraum liegen; daher muss ein vollständiger Algorithmus in der Lage sein, systematisch jeden Zustand zu erkunden, der vom Anfangszustand aus erreichbar ist. In endlichen Zustandsräumen ist das einfach zu erreichen: Solange wir die Pfade verfolgen und solche abschneiden, die Zyklen sind (z. B. von Arad nach Sibiu nach Arad), werden wir irgendwann jeden erreichbaren Zustand erreichen.

In unendlichen Zustandsräumen ist mehr Vorsicht geboten. Zum Beispiel würde ein Algorithmus, der wiederholt den Fakultätsoperator in Knuths „4“-Problem anwendet, einem unendlichen Pfad von 4 zu 4! zu (4!)! und so weiter folgen. In ähnlicher Weise folgt auf einem unendlichen Gitter ohne Hindernisse das wiederholte Vorwärtsbewegen in einer geraden Linie ebenfalls einem unendlichen Pfad neuer Zustände. In beiden Fällen kehrt der Algorithmus nie zu einem Zustand zurück, den er schon einmal erreicht hat, sondern ist unvollständig, weil weite Teile des Zustandsraums nie erreicht werden.

Damit ein Suchalgorithmus vollständig ist, muss er einen unendlichen Zustandsraum **systematisch** erforschen und sicherstellen, dass er schließlich jeden Zustand erreichen kann, der mit dem Anfangszustand verbunden ist. Auf dem unendlichen Gitter ist eine Art der systematischen Suche zum Beispiel ein spiralförmiger Pfad, der alle Zellen abdeckt, die s Schritte vom Ursprung entfernt sind, bevor er sich zu Zellen bewegt, die $s + 1$ Schritte entfernt sind. Leider muss ein guter Algorithmus in einem unendlichen Zustandsraum ohne Lösung ewig weitersuchen; er kann nicht abbrechen, weil er nicht wissen kann, ob der nächste Zustand ein Ziel sein wird.

Zeit- und Speicherplatzkomplexität werden im Hinblick auf ein Maß für die Schwierigkeit des Problems betrachtet. In der theoretischen Informatik ist das typische Maß die Größe des Zustandsraumgraphen, $|V| + |E|$, wobei $|V|$ die Anzahl der Ecken (Zustandsknoten, *vertice*) des Graphen ist und $|E|$ die Anzahl der Kanten (eindeutige Zustand-Aktion-Paare, *edge*). Dies ist passend, wenn der Graph eine explizite Datenstruktur ist, wie z. B. die Karte von Rumänien. Doch in vielen KI-Problemen wird der Graph nur *implizit* durch den Anfangszustand, die Aktionen und das Transitionsmodell beschrieben. Für einen impliziten Zustandsraum kann die Komplexität mithilfe der folgenden drei Größen gemessen werden: d, die **Tiefe** (*depth*) bzw. Anzahl der Aktionen in einer optimalen Lösung; m, die maximale Anzahl von Aktionen in einem beliebigen Pfad; und b, der **Verzweigungsfaktor** (*branching*) bzw. Anzahl von Nachfolgern eines Knotens, die betrachtet werden müssen.

3.4 Uninformierte Suchstrategien

Ein uninformierter Suchalgorithmus weiß nicht, wie nah ein Zustand am Ziel bzw. an den Zielen ist. Betrachten wir zum Beispiel unseren Agenten in Arad mit dem Ziel, Bukarest zu erreichen. Ein uninformierter Agent ohne Kenntnisse der rumänischen Geografie hat keine Ahnung, ob es besser ist, zuerst nach Zerind oder nach Sibiu zu fahren. Im Gegensatz dazu weiß ein informierter Agent (Abschnitt 3.5), der die Lage der einzelnen Städte kennt, dass Sibiu viel näher an Bukarest liegt und somit wahrscheinlich auf dem kürzesten Pfad liegt.

3.4.1 Breitensuche

Wenn alle Aktionen die gleichen Kosten haben, ist eine geeignete Strategie die **Breitensuche** (*Breadth-First Search*, BFS), bei der zuerst der Wurzelknoten expandiert wird, dann alle Nachfolger des Wurzelknotens, dann *deren* Nachfolger und so weiter. Dies ist eine systematische Suchstrategie, die daher auch auf unendlichen Zustandsräumen vollständig ist. Wir könnten die Breitensuche als einen Aufruf von BEST-FIRST-SEARCH implementieren, wobei die Evaluierungsfunktion $f(n)$ die Tiefe des Knotens ist – d. h. die Anzahl der Aktionen, die erforderlich sind, um den Knoten zu erreichen.

Wir können jedoch mit ein paar Tricks zusätzliche Effizienz erreichen. Eine First-In-First-Out-Warteschlange ist schneller als eine Prioritätswarteschlange und gibt uns die korrekte Reihenfolge der Knoten: Neue Knoten (die immer tiefer als ihre Eltern sind) wandern an das Ende der Warteschlange und alte Knoten, die flacher als die neuen Knoten sind, werden zuerst expandiert. Außerdem kann *reached* eine Menge von Zuständen sein und keine Abbildung von Zuständen auf Knoten, denn wenn wir einmal einen Zustand erreicht haben, können wir niemals einen besseren Pfad zu diesem Zustand finden. Das bedeutet auch, dass wir einen **frühen Zieltest** durchführen können, der prüft, ob ein Knoten eine Lösung ist, sobald er *erzeugt* wird, anstatt des **späten Zieltests**, den die Bestensuche verwendet, indem sie wartet, bis ein Knoten aus der Warteschlange entfernt wird. ► Abbildung 3.8 zeigt den Fortschritt einer Breitensuche in einem Binärbaum und ► Abbildung 3.9 zeigt den Algorithmus mit den Effizienzverbesserungen durch die frühen Zieltests.

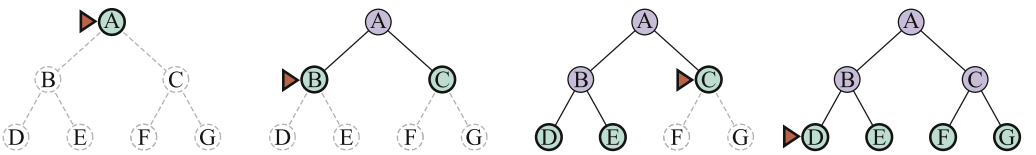


Abbildung 3.8: Breitensuche in einem einfachen Binärbaum. In jeder Phase wird der nächste zu erweiternde Knoten durch die dreieckige Markierung angezeigt.

```
function BREADTH-FIRST-SEARCH(problem) returns einen Knoten als Lösung oder failure
  node ← NODE(problem.INITIAL)
  if problem.IS-GOAL(node.STATE) then return node
  frontier ← eine FIFO-Warteschlagne mit node als einem Element
  reached ← {problem.INITIAL}
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if problem.IS-GOAL(s) then return child
      if s ist nicht in reached then
        füge s zu reached hinzu
        füge child zu frontier hinzu
  return failure

function UNIFORM-COST-SEARCH(problem) returns einen Knoten als Lösung oder failure
  return BEST-FIRST-SEARCH(problem, PATH-COST)
```

Abbildung 3.9: Algorithmen für Breitensuche und uniforme Kostensuche.

Die Breitensuche findet immer eine Lösung mit einer minimalen Anzahl von Aktionen, denn wenn sie Knoten in der Tiefe d erzeugt, hat sie bereits alle Knoten in der Tiefe $d-1$ generiert – falls einer von diesen eine Lösung ist, wäre diese also gefunden worden. Das heißt, die Suche ist kostenoptimal für Probleme, bei denen alle Aktionen die gleichen Kosten haben, jedoch nicht für Probleme, die diese Eigenschaft nicht haben. In beiden Fällen ist sie vollständig. Zum Zeit- und Platzbedarf: Stellen Sie sich vor, Sie durchsuchen einen uniformen Baum, bei dem jeder Zustand b Nachfolger hat. Die Wurzel des Suchbaums erzeugt b Knoten, von denen jeder b weitere Knoten erzeugt, insgesamt gibt es also b^2 Knoten auf der zweiten Ebene. Jeder von diesen erzeugt b weitere Knoten, was b^3 Knoten auf der dritten Ebene ergibt, und so weiter. Nehmen wir nun an, die Lösung befindet sich in der Tiefe d. Dann ist die Gesamtzahl der erzeugten Knoten:

$$1 + b + b^2 + b^3 + \cdots + b^d = O(b^d)$$

Alle Knoten bleiben im Speicher, sodass sowohl die Zeit- als auch die Speicherplatzkomplexität $O(b^d)$ ist. Solche exponentiellen Schranken sind erschreckend. Als typisches Beispiel aus der Praxis betrachten wir ein Problem mit dem Verzweigungsfaktor $b = 10$, einer Verarbeitungsgeschwindigkeit von 1 Million Knoten/Sekunde und einem Speicherbedarf von 1 KByte/Knoten. Eine Suche bis zur Tiefe $d = 10$ würde weniger als 3 Stunden dauern, aber 10 Terabyte Speicher benötigen. *Der Speicherbedarf ist für die Breitensuche ein größeres Problem als die Ausführungszeit.* Doch die Zeit ist immer noch ein wichtiger Faktor. Bei Tiefe $d = 14$ würde die Suche selbst bei unendlichem Speicher 3,5 Jahre dauern. Generell gilt: *Suchprobleme mit exponentieller Komplexität können außer für die kleinsten Instanzen nicht durch uninformierte Suche gelöst werden.*

3.4.2 Der Algorithmus von Dijkstra oder uniforme Kostensuche

Wenn Aktionen unterschiedliche Kosten haben, ist es naheliegend, die Bestensuche zu verwenden, bei der die Evaluierungsfunktion die Kosten des Pfads von der Wurzel zum aktuellen Knoten ist. Diese Suche wird in der theoretischen Informatik als Algorithmus von Dijkstra bezeichnet, in der KI-Gemeinschaft als **uniforme Kostensuche**. Während sich die Breitensuche in Wellen mit einheitlicher Tiefe ausbreitet – zuerst Tiefe 1, dann Tiefe 2 und so weiter –, ist hier der Grundgedanke, dass sich die uniforme Kostensuche in Wellen mit einheitlichen Pfadkosten ausbreitet. Der Algorithmus kann als Aufruf von BEST-FIRST-SEARCH mit PATH-COST als Evaluierungsfunktion implementiert werden, wie in ► Abbildung 3.9 gezeigt.

Betrachten Sie ► Abbildung 3.10, wo das Problem darin besteht, von Sibiu nach Bukarest zu kommen. Die Nachfolger von Sibiu sind Rimnicu Vilcea und Fagaras, mit Kosten von 80

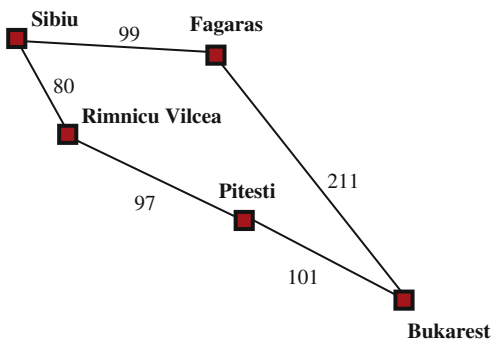


Abbildung 3.10: Ausgewählter Teil des Rumänien-Zustandsraums, um die uniforme Kostensuche zu veranschaulichen.

bzw. 99. Der Knoten mit den geringsten Kosten, Rimnicu Vilcea, wird als Nächstes expandiert, indem Pitesti mit Kosten von $80 + 97 = 177$ hinzugefügt wird. Der Knoten mit den geringsten Kosten ist nun Fagaras, also wird dieser expandiert und Bukarest wird mit Kosten $99 + 211 = 310$ hinzugefügt. Bukarest ist das Ziel, doch der Algorithmus testet nur beim Expandieren eines Knotens auf Ziele, nicht beim Erzeugen eines Knotens, also hat er noch nicht erkannt, dass dies ein Pfad zum Ziel ist.

Der Algorithmus arbeitet weiter, wählt als Nächstes Pitesti für die Expansion und fügt einen zweiten Pfad nach Bukarest mit den Kosten $80 + 97 + 101 = 278$ hinzu. Dieser Pfad hat niedrigere Kosten, also ersetzt er den vorhergehenden Pfad in *reached* und wird zu *frontier* hinzugefügt. Es stellt sich heraus, dass dieser Knoten nun die niedrigsten Kosten hat, also wird er als Nächster betrachtet, als Ziel erkannt und zurückgegeben. Beachten Sie, dass wir einen Pfad mit höheren Kosten (den durch Fagaras) zurückgegeben hätten, wenn wir beim Erzeugen eines Knotens und nicht beim Expandieren des Knotens mit den niedrigsten Kosten auf Erreichen des Ziels geprüft hätten.

Die Komplexität der uniformen Kostensuche wird charakterisiert durch C^*, die Kosten der optimalen Lösung,[8] und ϵ, einer unteren Schranke für die Kosten jeder Aktion, wobei $\epsilon > 0$ ist. Die Zeit- und Speicherplatzkomplexität des Algorithmus beträgt also im schlechtesten Fall $O(b^{1+\lfloor C^*/\epsilon \rfloor})$, was viel größer als b^d sein kann. Das liegt daran, dass die uniforme Kostensuche große Bäume von Aktionen mit niedrigen Kosten untersuchen kann, bevor sie Pfade erkundet, die eine teure und vielleicht nützliche Aktion beinhalten. Sind alle Aktionskosten gleich, dann ist $b^{1+\lfloor C^*/\epsilon \rfloor}$ einfach b^{d+1} und die uniforme Kostensuche ist mit der Breitensuche vergleichbar.

Die uniforme Kostensuche ist vollständig und kostenoptimal, weil die erste gefundene Lösung Kosten hat, die mindestens so niedrig sind wie die Kosten jedes anderen Knotens der Grenze. Die Suche betrachtet alle Pfade systematisch in der Reihenfolge steigender Kosten und läuft nie Gefahr, sich auf einen einzelnen unendlichen Pfad zu begeben (unter der Annahme, dass alle Aktionskosten $> \epsilon > 0$ sind).

3.4.3 Tiefensuche und das Speicherproblem

Bei der **Tiefensuche** (*Depth-First Search*, DFS) wird immer zuerst der *tiefste* Knoten der Grenze expandiert. Sie könnte als Aufruf von BEST-FIRST-SEARCH implementiert werden, wobei die Evaluierungsfunktion f das Negativ der Tiefe ist. Sie wird jedoch normalerweise nicht als Graphensuche, sondern als baumartige Suche implementiert, bei der keine Tabellen der erreichten Zustände geführt werden. Der Verlauf der Suche ist in ► Abbildung 3.11 dargestellt; die Suche geht sofort bis zur tiefsten Ebene des Suchbaums, wo die Knoten keine Nachfolger haben. Die Suche geht dann „rückwärts“ zum nächsttieferen Knoten, der noch nicht expandierte Nachfolger hat. Die Tiefensuche ist nicht kostenoptimal; sie gibt die erste gefundene Lösung zurück, auch wenn diese nicht die günstigste ist.

Für endliche Zustandsräume in Form von Bäumen ist die Tiefensuche effizient und vollständig; für azyklische Zustandsräume kann es passieren, dass am Ende derselbe Zustand viele Male über verschiedene Pfade expandiert wurde, aber (irgendwann) wird der gesamte Raum systematisch erkundet sein.

In zyklischen Zustandsräumen kann sich die Suche in einer Endlosschleife verfangen; daher überprüfen einige Implementierungen der Tiefensuche jeden neuen Knoten auf Zyklen. In unendlichen Zustandsräumen schließlich ist die Tiefensuche nicht systematisch: Sie kann auf einem unendlichen Pfad hängenbleiben, auch wenn es keine Zyklen gibt. Daher ist die Tiefensuche unvollständig.

8 Hier und im gesamten Buch bedeutet das Sternchen in C^* einen optimalen Wert für C.

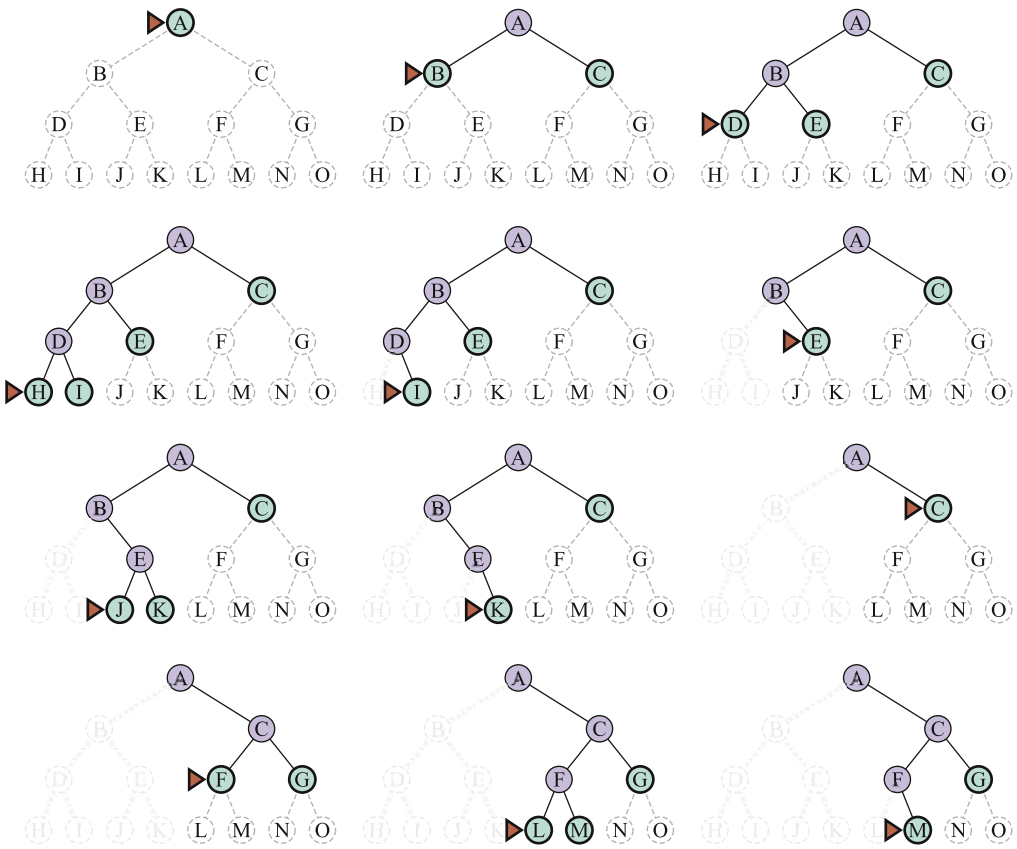


Abbildung 3.11: Dutzend Schritte (von links nach rechts, von oben nach unten) im Verlauf einer Tiefensuche in einem Binärbaum vom Startzustand A zu Ziel M. Die Grenze ist grün gezeichnet, wobei eine dreieckige Markierung den als Nächsten zu expandierenden Knoten kennzeichnet. Zuvor expandierte Knoten sind lavendelfarben und potenzielle zukünftige Knoten haben schwach gestrichelte Linien. Expandierte Knoten, die keine Nachkommen im Grenzbereich haben (sehr schwache Linien), können verworfen werden.

Warum sollte jemand bei all diesen schlechten Nachrichten eine Tiefensuche anstelle einer Breitensuche oder Bestensuche in Betracht ziehen? Die Antwort ist, dass bei Problemen, bei denen eine baumartige Suche möglich ist, die Tiefensuche einen viel geringeren Speicherbedarf hat. Wir führen erst gar keine *reached*-Tabelle und die Grenze ist sehr klein: Stellen Sie sich die Grenze bei der Breitensuche als die Oberfläche einer immer größer werdenden Kugel vor, während die Grenze bei der Tiefensuche nur der Radius der Kugel ist.

Für einen endlichen baumförmigen Zustandsraum wie in ► Abbildung 3.11 benötigt eine baumartige Tiefensuche Zeit, die proportional zur Anzahl der Zustände ist, und hat eine Speicherplatzkomplexität von nur $O(bm)$, wobei b der Verzweigungsfaktor und m die maximale Tiefe des Baums ist. Einige Probleme, die bei der Breitensuche Exabytes an Speicher benötigen würden, kommen bei der Tiefensuche mit nur einigen Kilobytes aus. Aufgrund der sparsamen Speichernutzung wurde die baumartige Tiefensuche als grundlegendes Arbeitspferd für viele KI-Bereiche übernommen, einschließlich der Constraint-Satisfaction-Probleme (Kapitel 6), de aussagenlogischen Erfüllbarkeit (Kapitel 7) und der logischen Programmierung (Kapitel 9).

Eine Variante der Tiefensuche, die **Backtracking-Suche**, benötigt sogar noch weniger Speicher (siehe Kapitel 6 für weitere Details). Beim Backtracking wird jeweils nur ein Nachfolger erzeugt und nicht alle Nachfolger; jeder partiell expandierte Knoten merkt sich, welcher Nachfolger als Nächstes erzeugt werden soll. Außerdem werden Nachfolger generiert, indem die aktuelle Zustandsbeschreibung direkt *geändert* wird, anstatt Speicher für einen ganz neuen Zustand vorzuhalten. Dies reduziert den Speicherbedarf auf nur eine Zustandsbeschreibung und einen Pfad von $O(m)$ Aktionen – eine signifikante Einsparung gegenüber $O(bm)$ Zuständen bei der Tiefensuche. Mit Backtracking haben wir auch die Möglichkeit, eine effiziente Mengendatenstruktur für die Zustände auf dem aktuellen Pfad zu verwalten, wodurch wir in $O(1)$ Zeit statt $O(m)$ nach einem zyklischen Pfad suchen können. Damit Backtracking funktioniert, müssen wir in der Lage sein, jede Aktion *rückgängig zu machen*, wenn wir zurückgehen. Backtracking ist entscheidend für den Erfolg vieler Probleme mit großen Zustandsbeschreibungen, wie z. B. bei der Robotermontage.

3.4.4 Beschränkte und iterative Tiefensuche

Um zu verhindern, dass sich die Tiefensuche auf einem unendlichen Pfad verliert, können wir die **beschränkte Tiefensuche** (*Depth-Limited Search*) verwenden, eine Version der Tiefensuche, bei der wir eine Tiefengrenze ℓ angeben und alle Knoten der Tiefe ℓ so behandeln, als hätten sie keine Nachfolger (▶ Abbildung 3.12). Die Zeitkomplexität ist $O(b^\ell)$ und die Speicherplatzkomplexität ist $O(b\ell)$. Treffen wir eine schlechte Wahl für ℓ, so wird der Algorithmus die Lösung leider nicht erreichen, wodurch er wieder unvollständig wird.

Da die Tiefensuche eine baumartige Suche ist, können wir nicht verhindern, dass sie generell Zeit auf redundanten Pfaden verschwendet, doch wir können Zyklen auf Kosten von etwas Rechenzeit eliminieren. Wenn wir nur ein paar Glieder in der Elternkette nach oben schauen, können wir die meisten Zyklen abfangen; längere Zyklen werden durch die Tiefenbegrenzung behandelt.

```
function ITERATIVE-DEEPENING-SEARCH(problem) returns eine Lösung oder failure
  for depth = 0 to ∞ do
    result ← DEPTH-LIMITED-SEARCH(problem, depth)
    if result ≠ cutoff then return result

function DEPTH-LIMITED-SEARCH(problem, ℓ) returns ein Knoten oder failure oder cutoff
  frontier ← eine LIFO-Warteschlange (Stapel) mit NODE(problem.INITIAL) als ein Element
  result ← failure
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    if DEPTH(node) > ℓ then
      result ← cutoff
    else if not IS-CYCLE(node) do
      for each child in EXPAND(problem, node) do
        füge child zu frontier hinzu
  return result
```

Abbildung 3.12: Die iterative Tiefensuche wendet wiederholt eine beschränkte Tiefensuche mit wachsenden Grenzen an. Sie gibt einen von drei unterschiedlichen Werttypen zurück: entweder einen Lösungsknoten; oder *failure*, wenn alle Knoten erschöpfend untersucht sind und es nachweislich in keiner Tiefe eine Lösung gibt; oder *cutoff*, was heißt, dass es eine Lösung in einer Tiefe unterhalb von ℓ geben könnte. Dies ist ein baumartiger Suchalgorithmus, der die erreichten Zustände (*reached*) nicht aufzeichnet und daher viel weniger Speicher als die Bestensuche benötigt, aber das Risiko eingeht, denselben Zustand mehrmals auf verschiedenen Pfaden zu besuchen. Und wenn der IS-CYCLE-Test nicht *alle* Zyklen prüft, kann der Algorithmus in einer Schleife hängenbleiben.

Manchmal kann eine gute Tiefengrenze aus der Kenntnis des Problems abgeleitet werden. Zum Beispiel gibt es auf der Karte von Rumänien 20 Städte. Daher ist $\ell = 19$ eine gültige Grenze. Doch wenn wir die Karte sorgfältig studierten, würden wir herausfinden, dass jede Stadt von jeder anderen Stadt mit höchstens 9 Aktionen erreicht werden kann. Diese Zahl, bekannt als der **Durchmesser** des Zustandsraumgraphen, gibt uns eine bessere Tiefengrenze, die zu einer effizienteren beschränkten Tiefensuche führt. Für die meisten Probleme kennen wir jedoch erst eine gute Tiefengrenze, wenn wir das Problem gelöst haben.

Die **iterative Tiefensuche** (*Iterative Deepening Search*, IDS) löst das Problem der Auswahl eines guten Werts für ℓ, indem sie alle Werte ausprobiert – zuerst 0, dann 1, dann 2 usw. –, bis entweder eine Lösung gefunden wird oder die beschränkte Tiefensuche den *failure*-Wert anstelle des *cutoff*-Werts liefert. Der Algorithmus ist in ► Abbildung 3.12 dargestellt. Die iterative Tiefensuche kombiniert viele der Vorteile von Tiefen- und Breitensuche. Wie bei der Tiefensuche sind die Speicheranforderungen bescheiden: $O(bd)$, falls es eine Lösung gibt, oder $O(bm)$ auf endlichen Zustandsräumen ohne Lösung. Wie die Breitensuche ist die iterative Tiefensuche optimal für Probleme, bei denen alle Aktionen die gleichen Kosten haben, und ist vollständig auf endlichen azyklischen Zustandsräumen oder auf jedem endlichen Zustandsraum, wenn wir die Knoten auf dem gesamten Pfad nach oben auf Zyklen überprüfen.

Die Zeitkomplexität ist $O(b^d)$, falls es eine Lösung gibt, oder $O(b^m)$, wenn es keine gibt. Jede Iteration der iterativen Tiefensuche erzeugt ebenso wie die Breitensuche eine neue Ebene, doch bei der Breitensuche werden dazu alle Knoten im Speicher gespeichert, während bei der iterativen Tiefensuche die vorherigen Ebenen wiederholt werden, wodurch Speicher auf Kosten von mehr Zeit gespart wird. ► Abbildung 3.13 zeigt vier Iterationen der iterativen Tiefensuche in einem binären Suchbaum, wobei die Lösung bei der vierten Iteration gefunden wird.

Die iterative Tiefensuche mag verschwenderisch erscheinen, weil Zustände oben im Suchbaum mehrfach neu erzeugt werden. Doch bei vielen Zustandsräumen befinden sich die meisten Knoten in der untersten Ebene, sodass es nicht viel ausmacht, dass die oberen Ebenen wiederholt werden. Bei einer iterativen Tiefensuche werden die Knoten auf der untersten Ebene (Tiefe d) einmal erzeugt, die auf der zweiten Ebene von unten zweimal und so weiter, bis zu den Kindern der Wurzel, die d-mal erzeugt werden. Im schlechtesten Fall ist also die Gesamtzahl der erzeugten Knoten

$$N(\text{IDS}) = (d)b^1 + (d-1)b^2 + (d-2)b^3 + \cdots + b^d\,,$$

was eine Zeitkomplexität von $O(b^d)$ ergibt – asymptotisch die gleiche Komplexität wie bei der Breitensuche. Wenn zum Beispiel $b = 10$ und $d = 5$ ist, dann erhalten wir:

$$N(\text{IDS}) = 50 + 400 + 3.000 + 20.000 + 100.000 = 123.450$$
$$N(\text{BFS}) = 10 + 100 + 1.000 + 10.000 + 100.000 = 111.110$$

Falls Sie sich wirklich wegen der Wiederholung Sorgen machen, können Sie einen hybriden Ansatz verwenden, der eine Breitensuche ausführt, bis fast der gesamte verfügbare Speicher aufgebraucht ist, und dann mit einer iterativen Tiefensuche von allen Knoten der Grenze aus weiterarbeitet. *Im Allgemeinen ist die iterative Tiefensuche die bevorzugte uninformierte Suchmethode, wenn der Suchzustandsraum zu groß ist, um in den Speicher zu passen, und die Tiefe der Lösung nicht bekannt ist.*

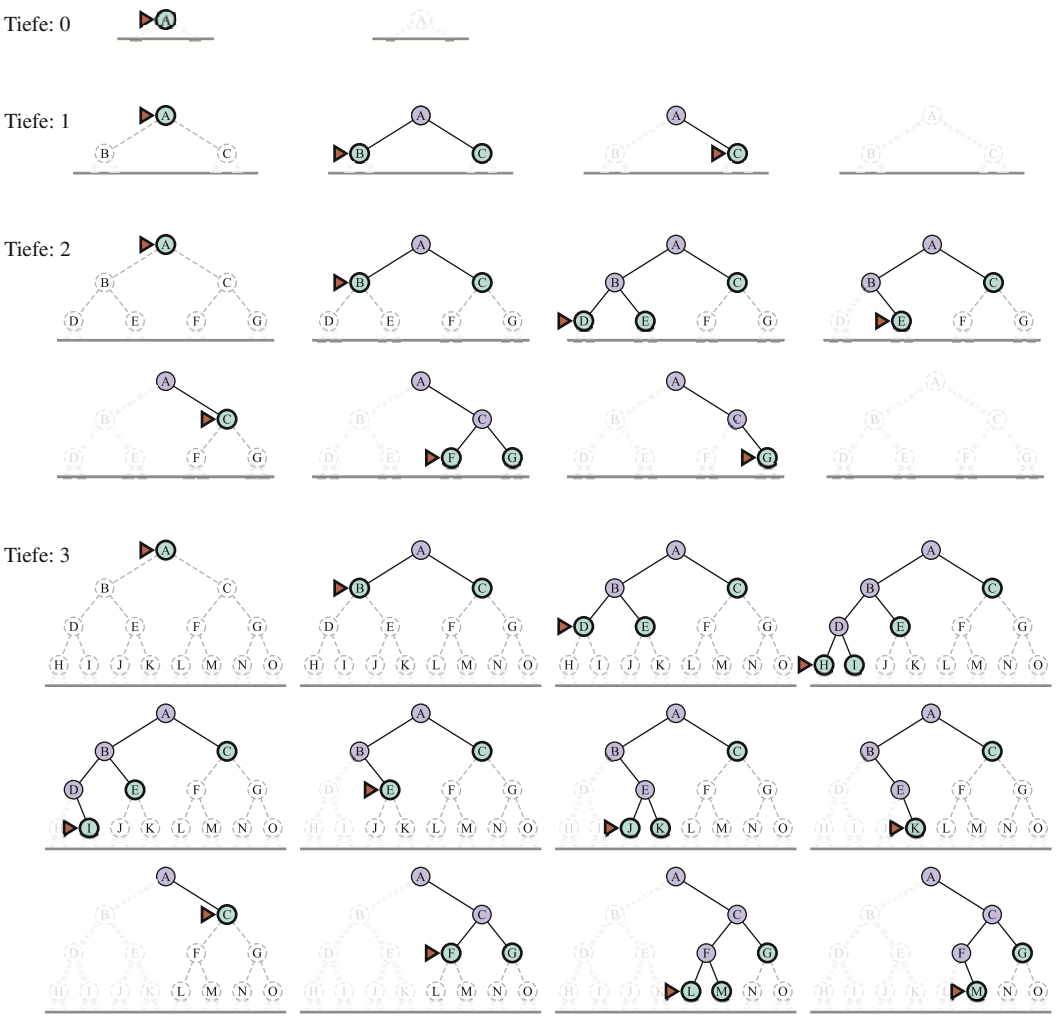


Abbildung 3.13: Vier Iterationen der iterativen Tiefensuche für das Ziel M in einem binären Baum, wobei die Tiefengrenze von 0 bis 3 variiert. Beachten Sie, dass die inneren Knoten einen einzelnen Pfad bilden. Das Dreieck markiert den als Nächstes zu expandierenden Knoten; grüne Knoten mit dunklen Umrissen gehören zur Grenze; die sehr dünn gezeichneten Knoten können nachweislich nicht Teil einer Lösung mit dieser Tiefengrenze sein.

3.4.5 Bidirektionale Suche

Die Algorithmen, die wir bisher behandelt haben, beginnen bei einem Anfangszustand und können einen von mehreren möglichen Zielzuständen erreichen. Ein alternativer Ansatz, der als **bidirektionale Suche** bezeichnet wird, sucht gleichzeitig vom Anfangszustand aus vorwärts und vom Zielzustand bzw. den Zielzuständen aus rückwärts – in der Hoffnung, dass sich die beiden Suchprozesse treffen. Die Grundidee dabei ist, dass $b^{d/2} + b^{d/2}$ viel kleiner ist als b^d (z. B. 50.000-mal kleiner, wenn $b = d = 10$).

Damit dies funktioniert, müssen wir zwei Grenzen und zwei Tabellen mit erreichten Zuständen im Auge behalten und wir müssen in der Lage sein, rückwärts zu denken: Wenn der Zustand s' ein Nachfolger von s in Vorwärtsrichtung ist, dann müssen wir wissen, dass s ein

Nachfolger von s' in Rückwärtsrichtung ist. Wir haben eine Lösung, wenn die beiden Grenzen aufeinanderstoßen.[9]

Es gibt viele verschiedene Versionen der bidirektionalen Suche, so wie es auch viele verschiedene unidirektionale Suchalgorithmen gibt. In diesem Abschnitt beschreiben wir die bidirektionale Bestensuche. Obwohl es zwei getrennte Grenzen gibt, ist der als Nächstes zu expandierende Knoten immer ein Knoten mit einem minimalen Wert der Auswertungsfunktion, was für beide Grenzen gilt. Wenn die Evaluierungsfunktion die Pfadkosten sind, erhalten wir eine bidirektionale uniforme Kostensuche, und wenn die Kosten des optimalen Pfads C^* sind, so wird kein Knoten mit Kosten $> \frac{C^*}{2}$ expandiert. Dies kann zu einer beträchtlichen Beschleunigung führen.

Der allgemeine bidirektionale Algorithmus für die Bestensuche ist in ► Abbildung 3.14 dargestellt. Wir übergeben dem Algorithmus jeweils zwei Versionen des Problems und der Evaluierungsfunktion, eine in Vorwärtsrichtung (tiefgestelltes F für *forward*) und eine in Rückwärtsrichtung (tiefgestelltes B für *backward*). Wenn die Evaluierungsfunktion die Pfadkosten sind, so wissen wir, dass die erste gefundene Lösung eine optimale Lösung sein wird, doch

9 In unserer Implementierung unterstützt die *reached*-Datenstruktur eine Abfrage, ob ein gegebener Zustand ein Element ist, und die *frontier*-Datenstruktur (eine Prioritätswarteschlange) tut dies nicht, sodass wir mithilfe von *reached* auf eine Kollision prüfen, allerdings fragen wir konzeptionell ab, ob sich die beiden Grenzen getroffen haben. Die Implementierung kann erweitert werden, um mehrere Zielzustände zu behandeln, indem der Knoten für jeden Zielzustand in die Tabellen $reached_B$ und $frontier_B$ geladen wird.

```
function BiBF-Search(problem_F, f_F, problem_B, f_B) returns eine Lösung oder failure
  node_F ← Node(problem_F.initial)                    // Knoten für einen Startzustand
  node_B ← Node(problem_B.initial)                    // Knoten für einen Zielzustand
  frontier_F ← eine Prioritätswarteschlange, sortiert durch f_F, mit node_F als einem Element
  frontier_B ← eine Prioritätswarteschlange, sortiert durch f_B, mit node_B als einem Element
  reached_F ← eine Nachschlagetabelle mit einem Schlüssel node_F.State und Wert node_F
  reached_B ← eine Nachschlagetabelle mit einem Schlüssel node_B.State und Wert node_B
  solution ← failure
  while not Terminated(solution, frontier_F, frontier_B) do
    if f_F(Top(frontier_F)) < f_B(Top(frontier_B)) then
      solution ← Proceed(F, problem_F, frontier_F, reached_F, reached_B, solution)
    else solution ← Proceed(B, problem_B, frontier_B, reached_B, reached_F, solution)
  return solution

function Proceed(dir, problem, frontier, reached, reached_2, solution) returns eine Lösung
        // Expandiert Knoten an der Grenze; vergleicht mit anderer Grenze in reached_2.
        // Die Variable „dir“ ist die Richtung: entweder F für „forward“ oder B für „backward“.
  node ← Pop(frontier)
  for each child in Expand(problem, node) do
    s ← child.State
    if s ist nicht in reached or Path-Cost(child) < Path-Cost(reached[s]) then
      reached[s] ← child
      füge child zu frontier hinzu
      if s ist in reached_2 then
        solution_2 ← Join-Nodes(dir, child, reached_2[s])
        if Path-Cost(solution_2) < Path-Cost(solution) then
          solution ← solution_2
  return solution
```

Abbildung 3.14: Die bidirektionale Bestensuche verwaltet zwei Grenzen und zwei Tabellen mit erreichten Zuständen. Wenn ein Pfad in einer Grenze einen Zustand erreicht, der auch in der anderen Hälfte der Suche erreicht wurde, werden die beiden Pfade vereinigt (durch die Funktion Join-Nodes) und bilden so eine Lösung. Es ist nicht garantiert, dass die erste erhaltene Lösung die beste ist – die Funktion Terminated bestimmt, wann die Suche nach neuen Lösungen beendet wird.

bei anderen Evaluierungsfunktionen ist das nicht unbedingt der Fall. Daher speichern wir die beste bisher gefundene Lösung und müssen diese eventuell mehrmals aktualisieren, bevor der TERMINATED-Test beweist, dass keine bessere Lösung mehr möglich ist.

3.4.6 Vergleich von uninformierten Suchalgorithmen

► Abbildung 3.15 vergleicht uninformierte Suchalgorithmen in Bezug auf die vier in Abschnitt 3.3.4 genannten Bewertungskriterien. Dieser Vergleich bezieht sich auf baumartige Suchversionen, die nicht auf wiederholte Zustände prüfen. Bei Graphen-Suchalgorithmen, die diese Prüfung vornehmen, bestehen die Hauptunterschiede darin, dass die Tiefensuche für endliche Zustandsräume vollständig ist und die Platz- und Zeitkomplexität durch die Größe des Zustandsraums (die Anzahl der Eckpunkte und Kanten, $|V| + |E|$) begrenzt ist.

Kriterium	Breitensuche	Uniforme Kostensuche	Tiefensuche	Beschränkte Tiefensuche	Iterative Tiefensuche	Bidirektional (falls möglich)
Vollständig?	Ja[1]	Ja[1,2]	Nein	Nein	Ja[1]	Ja[1,4]
Optimale Kosten?	Ja[3]	Ja	Nein	Nein	Ja[3]	Ja[3,4]
Zeit	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Platz	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$

Abbildung 3.15: Bewertung von Suchalgorithmen. b ist der Verzweigungsfaktor; m ist die maximale Tiefe des Suchbaums; d ist die Tiefe der flachsten Lösung bzw. m, wenn es keine Lösung gibt; ℓ ist die Tiefenbegrenzung. Die hochgestellten Symbole bedeuten: 1 – vollständig, falls b endlich ist und der Zustandsraum entweder eine Lösung hat oder endlich ist; 2 – vollständig, wenn alle Aktionskosten $\geq \epsilon > 0$ sind; 3 – kostenoptimal, falls alle Aktionskosten identisch sind; 4 – wenn beide Richtungen Breitensuchen oder uniforme Kostensuchen sind.

3.5 Informierte (heuristische) Suchstrategien

In diesem Abschnitt wird gezeigt, wie eine **informierte Suchstrategie** – die Wissen aus der Problemdomäne nutzt, um etwas über die Lage von Zielen zu erfahren – Lösungen effizienter finden kann als eine uninformierte Strategie. Das Wissen wird in Form einer **heuristischen Funktion** $h(n)$ zur Verfügung gestellt:[10]

$$h(n) = \text{geschätzte Kosten des günstigsten Pfads vom Zustand am Knoten } n \text{ zu einem Zielzustand}$$

Bei Wegfindungsproblemen können wir z. B. die Entfernung vom aktuellen Zustand zu einem Ziel schätzen, indem wir die Luftlinie auf der Karte zwischen den beiden Punkten berechnen. Wir untersuchen Heuristiken und deren Herkunft in Abschnitt 3.6 genauer.

3.5.1 Gierige Bestensuche

Die **gierige Bestensuche** (*Greedy Best-First Search*) ist eine Form der Bestensuche, bei der zuerst der Knoten mit dem niedrigsten $h(n)$-Wert expandiert wird – der Knoten, der dem Ziel

10 Es mag seltsam erscheinen, dass die heuristische Funktion einen Knoten als Eingabe verarbeitet, obwohl sie eigentlich nur den Zustand des Knotens benötigt. Es ist üblich, $h(n)$ statt $h(s)$ zu verwenden, um mit der Evaluierungsfunktion $f(n)$ und den Pfadkosten $g(n)$ konsistent zu bleiben.

Arad	366	**Mehadia**	241
Bukarest	0	**Neamt**	234
Craiova	160	**Oradea**	380
Drobeta	242	**Pitesti**	100
Eforie	161	**Rimnicu Vilcea**	193
Fagaras	176	**Sibiu**	253
Giurgiu	77	**Timisoara**	329
Hirsova	151	**Urziceni**	80
Iasi	226	**Vaslui**	199
Lugoj	244	**Zerind**	374

Abbildung 3.16: Werte für h_{LL} – Luftlinien nach Bukarest.

am nächsten zu sein scheint – mit dem Hintergrund, dass dies wahrscheinlich schnell zu einer Lösung führt. Die Evaluierungsfunktion ist damit $f(n) = h(n)$.

Schauen wir uns an, wie dies für Wegfindungsprobleme in Rumänien funktioniert; wir verwenden die Heuristik der **Luftlinie**, die wir h_{LL} nennen werden. Wenn das Ziel Bukarest ist, benötigen wir die Luftlinien nach Bukarest, die in ► Abbildung 3.16 angegeben werden. Zum Beispiel: $h_{LL}(Arad) = 366$. Beachten Sie, dass die Werte von h_{LL} nicht aus der Problembeschreibung selbst berechnet werden können (d. h. aus den Funktionen ACTIONS und RESULT). Außerdem benötigt man ein gewisses Maß an Weltwissen, um zu wissen, dass h_{LL} mit tatsächlichen Straßenentfernungen korreliert und daher eine sinnvolle Heuristik ist.

► Abbildung 3.17 zeigt den Fortschritt einer gierigen Bestensuche mit h_{LL}, um einen Pfad von Arad nach Bukarest zu finden. Der erste Knoten, der von Arad aus expandiert wird, ist Sibiu, da die Heuristik besagt, dass es näher an Bukarest liegt als Zerind oder Timisoara. Der nächste Knoten, der expandiert wird, ist Fagaras, weil dieser Ort jetzt laut Heuristik am nächsten liegt. Fagaras wiederum erzeugt Bukarest, welches das Ziel ist. Für dieses spezielle Problem findet die gierige Bestensuche mit h_{LL} eine Lösung, ohne je einen Knoten zu expandieren, der nicht auf dem Lösungspfad liegt. Die gefundene Lösung ist jedoch nicht kostenoptimal: Der Weg über Sibiu und Fagaras nach Bukarest ist 32 Meilen länger als der Weg über Rimnicu Vilcea und Pitesti. Aus diesem Grund wird der Algorithmus „gierig“ genannt – bei jeder Iteration versucht er, dem Ziel so nahe wie möglich zu kommen, aber Gier kann zu schlechteren Ergebnissen führen als vorsichtiges Vorgehen.

Die gierige Bestensuche in Graphen ist in endlichen Zustandsräumen vollständig, aber nicht in unendlichen. Die Zeit- und Speicherplatzkomplexität ist im schlechtesten Fall $O(|V|)$. Mit einer guten heuristischen Funktion kann die Komplexität jedoch erheblich reduziert werden und erreicht bei bestimmten Problemen $O(bm)$.

3.5.2 A*-Suche

Der gebräuchlichste informierte Suchalgorithmus ist die **A*-Suche** (gesprochen „A-Stern“), eine Bestensuche, die die Evaluierungsfunktion

$$f(n) = g(n) + h(n)$$

verwendet, wobei $g(n)$ die Pfadkosten vom Anfangszustand zum Knoten n und $h(n)$ die *geschätzten* Kosten des kürzesten Pfads von n zu einem Zielzustand sind, wir haben also:

$$f(n) = \text{ geschätzte Kosten des besten Pfads, der von } n \text{ zu einem Ziel führt}$$

In ► Abbildung 3.18 zeigen wir den Verlauf einer A*-Suche mit dem Ziel, Bukarest zu erreichen. Die Werte von g werden aus den Aktionskosten in ► Abbildung 3.1 berechnet, und die

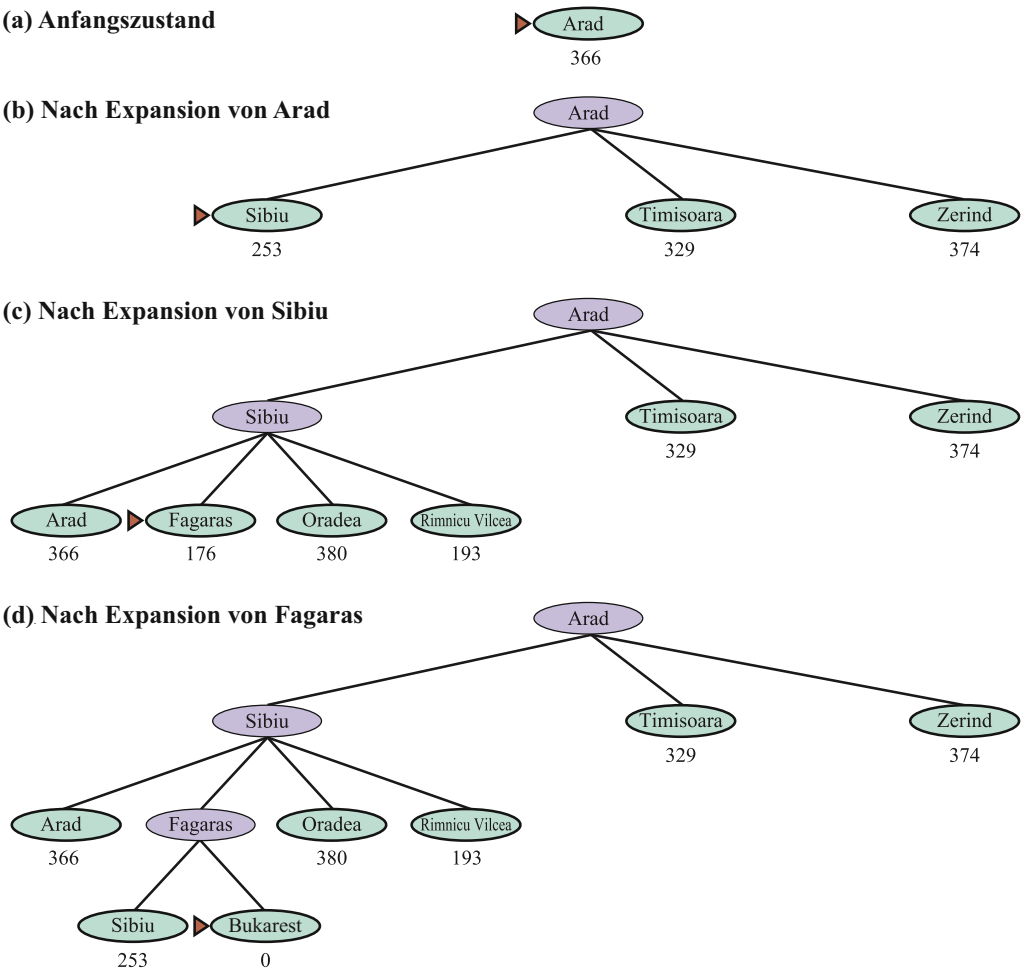


Abbildung 3.17: Phasen einer gierigen baumartigen Bestensuche nach Bukarest mit der Luftlinienheuristik h_{LL}. Die Knoten sind mit ihren h-Werten beschriftet.

Werte von h_{LL} sind in ► Abbildung 3.16 angegeben. Beachten Sie, dass Bukarest zum ersten Mal im Schritt (e) auf der Grenze erscheint, aber nicht für die Expansion ausgewählt (und somit nicht als Lösung erkannt) wird, weil Bukarest mit $f = 450$ nicht der Knoten mit den niedrigsten Kosten auf der Grenze ist – das wäre Pitesti mit $f = 417$. Man könnte auch sagen, dass es eine Lösung über Pitesti geben *könnte*, dessen Pfadkosten nur bei 417 liegen, daher wird sich der Algorithmus nicht mit einer Lösung von 450 zufrieden geben. In Schritt (f) ist nun ein anderer Pfad nach Bukarest der Knoten mit den niedrigsten Kosten mit $f = 418$, also wird dieser ausgewählt und als optimale Lösung erkannt.

Die A*-Suche ist vollständig.[11] Ob A* kostenoptimal ist, hängt von bestimmten Eigenschaften der Heuristik ab. Eine Schlüsseleigenschaft ist die **Zulässigkeit**: eine **zulässige Heuristik** ist eine Heuristik, die die Kosten zum Erreichen eines Ziels *niemals überschätzt*. (Eine zulässige Heuristik ist also *optimistisch*.) Mit einer zulässigen Heuristik ist A* kostenoptimal, was wir

11 Wiederum unter der Annahme, dass alle Aktionskosten $> \epsilon > 0$ sind und der Zustandsraum entweder eine Lösung hat oder endlich ist.

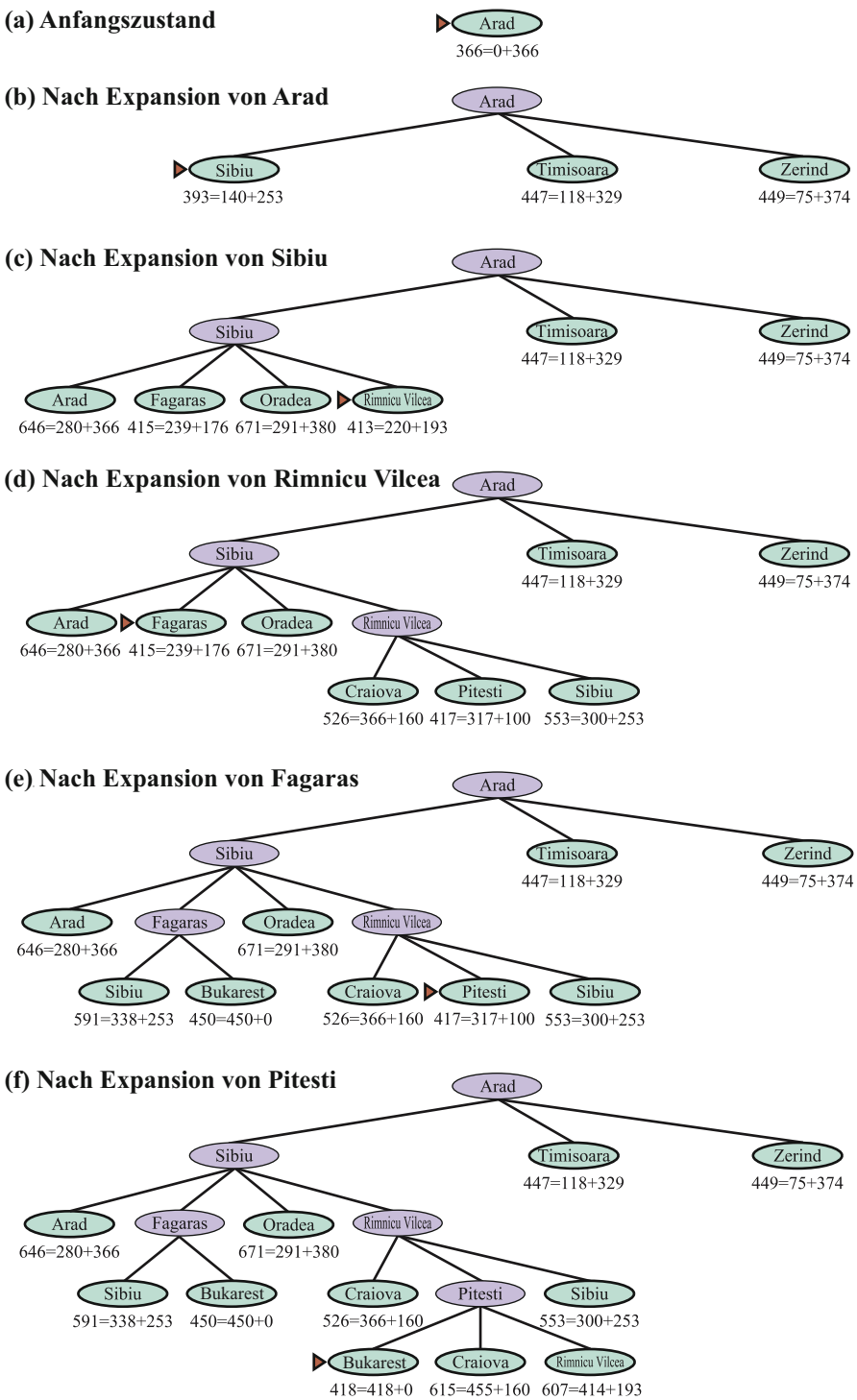


Abbildung 3.18: Phasen einer A*-Suche nach Bukarest. Knoten sind mit $f = g + h$ beschriftet. Die h-Werte sind die Luftlinien nach Bukarest aus Abbildung 3.16.

mit einem Widerspruchsbeweis zeigen können. Angenommen, der optimale Pfad hat Kosten C^*, doch der Algorithmus gibt einen Pfad mit Kosten $C > C^*$ zurück. Dann muss es einen Knoten n geben, der auf dem optimalen Pfad liegt und der nicht expandiert ist (denn falls alle Knoten auf dem optimalen Pfad expandiert wären, dann hätten wir bereits diese optimale Lösung zurückgegeben). Wenn wir also die Notation $g^*(n)$ verwenden, um die Kosten des optimalen Pfads vom Start bis n zu bezeichnen, und $h^*(n)$, um die Kosten des optimalen Pfads von n zum nächstgelegenen Ziel zu bezeichnen, dann erhalten wir:

$$
\begin{aligned}
&f(n) > C^* \quad \text{(andernfalls wäre } n \text{ expandiert)}\\
&f(n) = g(n) + h(n) \quad \text{(per Definition)}\\
&f(n) = g^*(n) + h(n) \quad \text{(weil } n \text{ auf einem optimalen Pfad ist)}\\
&f(n) \leq g^*(n) + h^*(n) \quad \text{(aufgrund der Zulässigkeit, } h(n) \leq h^*(n))\\
&f(n) \leq C^* \quad \text{(nach Definition, } C^* = g^*(n) + h^*(n))
\end{aligned}
$$

Die erste und die letzte Zeile bilden einen Widerspruch, also muss die Annahme, dass der Algorithmus einen suboptimalen Pfad zurückgeben könnte, falsch sein – A* gibt daher nur kostenoptimale Pfade zurück.

Eine etwas stärkere Eigenschaft wird als **Konsistenz** bezeichnet. Eine Heuristik $h(n)$ ist konsistent, falls für jeden Knoten n und jeden Nachfolger n' von n, der durch eine Aktion a erzeugt wird, gilt:

$$h(n) \leq c(n, a, n') + h(n')$$

Dies ist eine Form der **Dreiecksungleichung**, die besagt, dass eine Seite eines Dreiecks nicht länger sein kann als die Summe der beiden anderen Seiten (► Abbildung 3.19). Ein Beispiel für eine konsistente Heuristik ist die Luftlinie h_{LL}, die wir bei der Reise nach Bukarest verwendet haben.

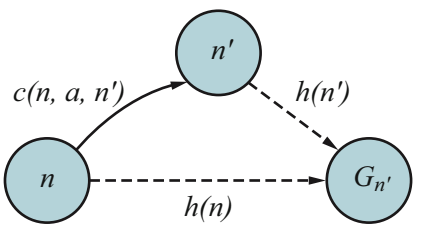


Abbildung 3.19: Dreiecksungleichung: Ist die Heuristik h **konsistent**, dann ist der einzelne Wert $h(n)$ kleiner als die Summe der Kosten $c(n, a, a')$ der Aktion von n bis n' plus der heuristischen Schätzung $h(n')$.

Jede konsistente Heuristik ist zulässig (aber nicht umgekehrt), also ist A* mit einer konsistenten Heuristik kostenoptimal. Außerdem gilt bei einer konsistenten Heuristik Folgendes: Wenn wir einen Zustand das erste Mal erreichen, befindet er sich auf einem optimalen Pfad, sodass wir nie einen Zustand neu zur Grenze hinzufügen müssen und nie einen Eintrag in *reached* ändern müssen. Doch mit einer inkonsistenten Heuristik könnten wir am Ende mehrere Pfade haben, die denselben Zustand erreichen, und wenn jeder neue Pfad niedrigere Pfadkosten hat als der vorherige, dann haben wir am Ende mehrere Knoten für diesen Zustand der Grenze, was uns sowohl Zeit als auch Platz kostet. Aus diesem Grund achten einige Implementierungen von A* darauf, einen Zustand nur einmal in die Grenze aufzunehmen, und falls ein besserer Pfad zum Zustand gefunden wird, werden alle Nachfolger des Zustands aktualisiert (was erfordert, dass Knoten sowohl Kind-Zeiger als auch Eltern-Zeiger haben). Diese Komplikationen haben dazu geführt, dass viele Implementierer inkonsistente Heuristiken vermeiden, doch Felner *et al.* (2011) führen an, dass die schlimmsten Effekte in der Praxis selten auftreten und man keine Angst vor inkonsistenten Heuristiken haben sollte.

Mit einer unzulässigen Heuristik kann A* kostenoptimal sein oder auch nicht. Hier sind zwei Fälle, in denen A* optimal ist: Erstens, wenn es auch nur einen kostenoptimalen Pfad gibt, auf dem $h(n)$ für alle Knoten n auf dem Pfad zulässig ist, dann wird dieser Pfad gefunden, unabhängig davon, wie die Heuristik die Zustände außerhalb des Pfads bewertet. Zweiter Fall: Wenn die optimale Lösung die Kosten C^* und die zweitbeste die Kosten C_2 hat und $h(n)$ einige Kosten überschätzt, aber nie um mehr als $C_2 - C^*$, dann liefert A* garantiert die kostenoptimalen Lösungen.

3.5.3 Suchkonturen

Eine gute Art, eine Suche zu visualisieren, ist das Zeichnen von **Konturen** im Zustandsraum, ähnlich den Konturen in einer topografischen Karte. ► Abbildung 3.20 zeigt ein Beispiel dafür. Innerhalb der Kontur, die mit 400 beschriftet ist, haben alle Knoten $f(n) = g(n)+h(n) \leq 400$ und so weiter. Da A* den Grenzknoten mit den niedrigsten f-Kosten expandiert, kann man sehen, dass sich eine A*-Suche vom Startknoten aus fächerförmig ausbreitet und Knoten in konzentrischen Bändern mit steigenden f-Kosten hinzufügt.

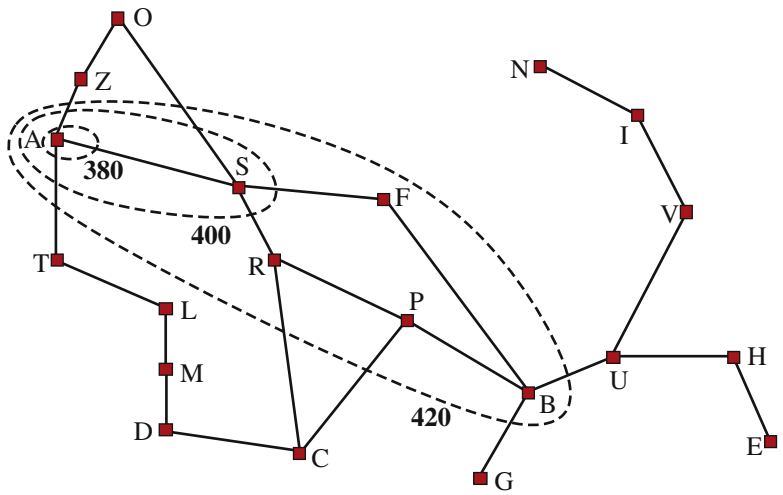


Abbildung 3.20: Karte von Rumänien mit Konturen für $f = 380$, $f = 400$ und $f = 420$ mit Arad als Startzustand. Knoten innerhalb einer bestimmten Kontur haben $f = g + h$-Kosten kleiner oder gleich dem Konturwert.

Bei der uniformen Kostensuche sehen wir ebenfalls Konturen, doch mit g-Kosten anstatt $g+h$. Die Konturen bei der uniformen Kostensuche sind „kreisförmig“ um den Startzustand herum und breiten sich gleichmäßig in alle Richtungen aus, ohne Präferenz in Richtung des Ziels. Bei der A*-Suche mit einer guten Heuristik dehnen sich die $g+h$-Bänder in Richtung eines Zielzustands aus (wie in ► Abbildung 3.20) und konzentrieren sich enger um einen optimalen Pfad.

Es sollte klar sein, dass die g-Kosten **monoton** sind, wenn Sie einen Pfad verlängern: Die Pfadkosten steigen immer, wenn Sie einen Pfad entlang gehen, weil die Aktionskosten immer positiv sind.[12] Daher erhalten Sie konzentrische Konturlinien, die sich nicht kreuzen, und wenn Sie sich entscheiden, die Linien fein genug zu zeichnen, können Sie eine Linie zwischen zwei beliebigen Knoten auf jedem Pfad ziehen.

12 Technisch gesehen sagen wir „streng monoton“ bei Kosten, die immer steigen, und „monoton“ bei Kosten, die nie sinken, aber gleich bleiben können.

Doch es ist nicht offensichtlich, ob die Kosten für $f = g + h$ monoton ansteigen werden. Wenn Sie einen Pfad von n nach n' verlängern, gehen die Kosten von $g(n) + h(n)$ zu $g(n) + c(n, a, n') + h(n')$. Streichen wir den Term $g(n)$, so sehen wir, dass die Kosten des Pfads genau dann monoton ansteigen, wenn $h(n) \leq c(n, a, n') + h(n')$ ist – mit anderen Worten, genau dann, wenn die Heuristik konsistent ist.[13] Beachten Sie aber, dass ein Pfad mehrere Knoten hintereinander mit dem gleichen $g(n)+h(n)$-Wert beitragen kann; dies geschieht immer dann, wenn die Verringerung von h genau den soeben durchgeführten Aktionskosten entspricht (z. B. in einem Gitterproblem, wenn n in der gleichen Reihe wie das Ziel liegt und Sie einen Schritt in Richtung des Ziels machen, wird g um 1 erhöht und h um 1 verringert). Sind C^* die Kosten des optimalen Lösungswegs, dann können wir Folgendes festhalten:

- A* expandiert alle Knoten, die vom Anfangszustand aus auf einem Pfad erreicht werden können, wobei für jeden Knoten auf dem Pfad $f(n) < C^*$ gilt. Wir sagen, dass dies **sicher expandierte Knoten** sind.
- A* könnte dann einige der Knoten direkt auf der „Zielkontur“ (mit $f(n) = C^*$) expandieren, bevor ein Zielknoten ausgewählt wird.
- A* expandiert keine Knoten mit $f(n) > C^*$.

Wir sagen, dass A* mit einer konsistenten Heuristik **optimal effizient** in dem Sinne ist, dass jeder Algorithmus, der Suchpfade vom Anfangszustand aus erweitert und dieselben heuristischen Informationen verwendet, alle Knoten expandieren muss, die von A* sicher expandiert werden (weil jeder von ihnen Teil einer optimalen Lösung gewesen sein könnte). Unter den Knoten mit $f(n) = C^*$ könnte ein Algorithmus Glück haben und den optimalen Knoten zuerst auswählen, während ein anderer Algorithmus Pech hat – diesen Unterschied berücksichtigen wir bei der Definition der optimalen Effizienz nicht.

A* ist effizient, weil es Suchbaumknoten **kürzt** (*Pruning*), die für das Finden einer optimalen Lösung nicht notwendig sind. In ▶ Abbildung 3.18 b sehen wir, dass für Timisoara $f = 447$ und für Zerind $f = 449$ gilt. Obwohl beides Kindknoten der Wurzel sind und zu den ersten Knoten gehören würden, die bei der uniformen Kosten- oder Breitensuche expandiert werden, werden sie bei der A*-Suche nie expandiert, weil die Lösung mit $f = 418$ zuerst gefunden wird. Das Konzept des Kürzens – das Eliminieren von Möglichkeiten auf Basis des vorhandenen Wissen, ohne dies näher untersuchen zu müssen – ist für viele Bereiche der KI wichtig.

Die A*-Suche ist also unter all diesen Algorithmen vollständig, kostenoptimal und optimal effizient – was recht zufriedenstellend ist, doch leider heißt es noch nicht, dass A* die Antwort auf alle unsere Suchanforderungen ist. Der Haken dabei ist, dass bei vielen Problemen die Anzahl der expandierten Knoten exponentiell zur Länge der Lösung wachsen kann. Betrachten wir zum Beispiel eine Version der Staubsaugerwelt mit einem superstarken Staubsauger, der jedes beliebige Feld mit einem Aufwand von 1 Einheit säubern kann, ohne das Feld überhaupt besuchen zu müssen – in diesem Szenario können die Felder in beliebiger Reihenfolge gereinigt werden. Bei N anfänglich verschmutzten Feldern gibt es 2^N Zustände, in denen eine Teilmenge gereinigt wurde; alle diese Zustände liegen auf einem optimalen Lösungsweg und erfüllen daher $f(n) < C^*$, sodass alle von A* besucht würden.

3.5.4 Satisficing-Suche: Unzulässige Heuristiken und gewichtete A*-Algorithmen

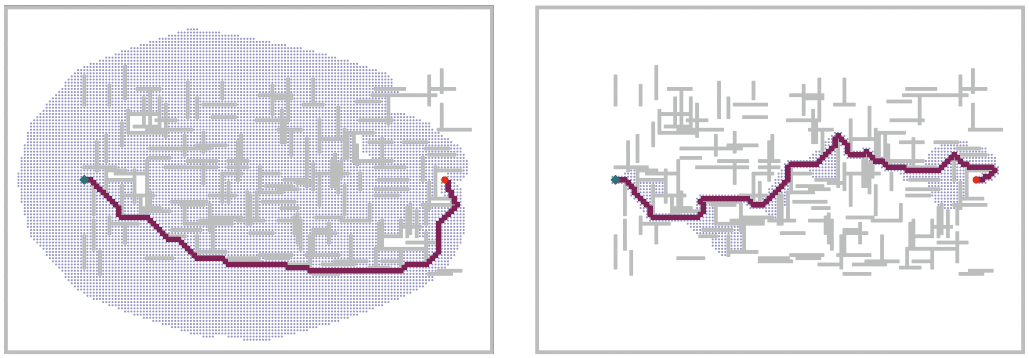
Die A*-Suche hat viele gute Eigenschaften, doch sie expandiert viele Knoten. Wir können weniger Knoten erforschen (und damit weniger Zeit und Platz verbrauchen), wenn wir bereit sind, Lösungen zu akzeptieren, die zwar suboptimal, aber „gut genug“ sind – was wir als

13 In der Tat ist der Begriff „monotone Heuristik“ ein Synonym für „konsistente Heuristik“. Die beiden Konzepte wurden unabhängig voneinander entwickelt, dann wurde bewiesen, dass sie äquivalent sind (Pearl, 1984).

zufriedenstellende (*satisficing*) Lösungen bezeichnen. Wenn wir der A*-Suche erlauben, eine unzulässige Heuristik zu verwenden – eine, die eventuell überschätzt – dann riskieren wir, die optimale Lösung zu verpassen, aber die Heuristik ist möglicherweise genauer, was die Anzahl der expandierten Knoten reduziert. Straßenbauingenieure kennen zum Beispiel das Konzept eines **Umwegindexes**, ein Wert, der mit der Luftlinie multipliziert wird, um die typische Krümmung von Straßen zu berücksichtigen. Ein Umwegindex von 1,3 bedeutet, dass für zwei Städte, die 10 km Luftline voneinander entfernt sind, eine gute Schätzung für den besten Pfad zwischen ihnen 13 km beträgt. Für die meisten Orte liegt der Umwegindex zwischen 1,2 und 1,6.

Wir können dieses Konzept auf jedes beliebige Problem anwenden, nicht nur auf solche, bei denen es um Straßen geht, und zwar mit einem Ansatz, der **gewichtete A*-Suche** genannt wird, bei der wir den heuristischen Wert stärker gewichten, sodass wir die Evaluierungsfunktion $f(n) = g(n) + W \times h(n)$ erhalten, für irgendein $W > 1$.

► Abbildung 3.21 zeigt ein Suchproblem in einer Gitterwelt. In ► Abbildung 3.21 a findet eine A*-Suche die optimale Lösung, muss dafür aber einen großen Teil des Zustandsraums erkunden. In ► Abbildung 3.21 b findet eine gewichtete A*-Suche eine Lösung, die etwas teurer, doch viel schneller ist. Wir sehen, dass die gewichtete Suche die Kontur der erreichten Zustände auf ein Ziel konzentriert. Das bedeutet, dass weniger Zustände erforscht werden, doch wenn der optimale Pfad außerhalb der Kontur der gewichteten Suche liegt (wie in diesem Fall), dann wird er nicht gefunden. Kostet die optimale Lösung C^*, so wird eine gewichtete A*-Suche im Allgemeinen eine Lösung finden, die irgendwo zwischen C^* und $W \times C^*$ liegt; in der Praxis erhalten wir jedoch normalerweise Ergebnisse, die viel näher an C^* als an $W \times C^*$ liegen.



(a) (b)

Abbildung 3.21: Zwei Suchen auf demselben Gitter: (a) eine A*-Suche und (b) eine gewichtete A*-Suche mit Gewicht $W = 2$. Die grauen Balken sind Hindernisse, die violette Linie ist der Weg vom grünen Start zum roten Ziel und die kleinen Punkte sind Zustände, die von jeder Suche erreicht wurden. Bei diesem speziellen Problem erkundet die gewichtete A*-Suche 7-mal weniger Zustände und findet einen Pfad, der 5 % teurer ist.

Wir haben Suchen betrachtet, die Zustände durch Kombination von g und h auf verschiedene Weise bewerten; die gewichtete A*-Suche kann als eine Verallgemeinerung der anderen Suchmethoden betrachtet werden:

$$\begin{aligned} \text{A*-Suche:} &\quad g(n) + h(n) (W = 1) \\ \text{uniforme Kostensuche:} &\quad g(n) (W = 0) \\ \text{gierige Bestensuche:} &\quad h(n) (W = \infty) \\ \text{gewichtete A*-Suche:} &\quad g(n) + W \times h(n) (1 < W < \infty) \end{aligned}$$

Man könnte die gewichtete A*-Suche als „ein bisschen gierig“ bezeichnen: wie die gierige Bestensuche konzentriert sie sich auf ein Ziel; andererseits ignoriert sie die Pfadkosten nicht vollständig und stellt einen Pfad zurück, der wenig Fortschritt zu großen Kosten bringt.

Es gibt eine Vielzahl von suboptimalen Suchalgorithmen, die durch die Kriterien dafür, was als „gut genug“ gilt, charakterisiert werden können. Bei der **beschränkten suboptimalen Suche** suchen wir nach einer Lösung, die garantiert innerhalb eines konstanten Faktors W der optimalen Kosten liegt. Die gewichtete A*-Suche bietet diese Garantie. Bei der **Suche mit beschränkten Kosten** suchen wir nach einer Lösung, deren Kosten kleiner sind als eine Konstante C. Und bei der **Suche ohne beschränkte Kosten** akzeptieren wir eine Lösung mit beliebigen Kosten, solange wir sie schnell finden können.

Ein Beispiel für einen Suchalgorithmus mit unbeschränkten Kosten ist die **Speedy-Suche**, die eine Version der gierigen Bestensuche ist und als Heuristik die geschätzte Anzahl der zum Erreichen eines Ziels erforderlichen Aktionen verwendet, unabhängig von den Kosten dieser Aktionen. Für Probleme, bei denen alle Aktionen die gleichen Kosten haben, ist die Speedy-Suche also dieselbe wie die gierige Bestensuche, doch wenn die Aktionen unterschiedliche Kosten haben, findet sie in der Regel schnell eine Lösung, auch wenn diese vielleicht hohe Kosten hat.

3.5.5 Speicherbeschränkte Suche

Das Hauptproblem von A* ist der Speicherverbrauch. In diesem Abschnitt werden wir einige Implementierungstricks behandeln, die Platz sparen, sowie einige völlig neue Algorithmen, die den verfügbaren Platz besser ausnutzen.

Der Speicher wird zwischen den Zuständen der Grenze (*frontier*) und den erreichten Zuständen (*reached*) aufgeteilt. In unserer Implementierung der Bestensuche wird ein Zustand, der sich auf der Grenze befindet, an zwei Stellen gespeichert: als Knoten in der Grenze (damit wir entscheiden können, was als Nächstes expandiert werden soll) und als Eintrag in der Tabelle der erreichten Zustände (damit wir wissen, ob wir den Zustand schon einmal besucht haben). Für viele Probleme (z. B. die Erkundung eines Gitters) ist diese Duplizierung kein Problem, da die Größe von *frontier* viel kleiner ist als die von *reached*, sodass die Duplizierung der Zustände in der Grenze eine vergleichsweise triviale Menge an Speicher erfordert. Doch einige Implementierungen speichern einen Zustand nur an einem der beiden Orte, was ein wenig Speicherplatz einspart, doch den Algorithmus komplizierter (und vielleicht langsamer) macht.

Eine andere Möglichkeit ist, Zustände aus *reached* zu entfernen, wenn wir beweisen können, dass sie nicht mehr benötigt werden. Für einige Probleme können wir die Trennungseigenschaft (► Abbildung 3.6 auf S. 102) zusammen mit dem Verbot von Kehrtwenden benutzen, um sicherzustellen, dass sich alle Aktionen entweder von der Grenze weg- oder auf einen anderen Grenzzustand zubewegen. In diesem Fall müssen wir nur die Grenze auf redundante Pfade überprüfen und brauchen die *reached*-Tabelle nicht mehr.

Bei anderen Problemen können wir **Referenzzähler** verwenden, die aufzeichnen, wie oft ein Zustand erreicht wurde, und diesen aus der *reached*-Tabelle entfernen, wenn es keine weiteren Möglichkeiten gibt, den Zustand zu erreichen. In einer Gitterwelt, in der jeder Zustand nur von seinen vier Nachbarn aus erreicht werden kann, können wir beispielsweise einen Zustand aus der Tabelle entfernen, sobald er viermal erreicht wurde.

Betrachten wir nun neue Algorithmen, die auf eine sparsame Speichernutzung ausgelegt sind.

Die **Strahlsuche** (*Beam Search*) begrenzt die Größe der Grenze. Der einfachste Ansatz ist, nur die k Knoten mit den besten f-Werten zu behalten und alle anderen expandierten Knoten zu verwerfen. Dadurch wird die Suche natürlich unvollständig und suboptimal, aber wir können

k so wählen, dass der verfügbare Speicher gut genutzt wird, außerdem wird der Algorithmus schnell ausgeführt, da er weniger Knoten expandiert. Für viele Probleme kann er gute, nahezu optimale Lösungen finden. Man kann sich die uniforme Kosten- oder A*-Suche so vorstellen, dass sie sich überall in konzentrischen Konturen ausbreitet, und man kann sich die Strahlsuche so vorstellen, dass sie nur einen fokussierten Teil dieser Konturen erforscht – den Teil, der die k besten Kandidaten enthält.

Eine alternative Version der Strahlsuche beschränkt die Größe der Grenze weniger stark, sondern speichert stattdessen jeden Knoten, dessen f-Wert innerhalb von δ des besten f-Werts liegt. Auf diese Weise werden, wenn es einige Knoten mit starken Werten gibt, nur ein paar behalten, doch wenn es keine starken Knoten gibt, werden mehr gespeichert, bis ein starker Knoten auftaucht.

Die **iterative A*-Tiefensuche** (*Iterative Deepening A**, IDA*) ist für A* das, was die iterative Tiefensuche für die allgemeine Tiefensuche ist: IDA* bietet uns die Vorteile von A*, ohne dass alle erreichten Zustände im Speicher gehalten werden müssen, allerdings zu dem Preis, dass einige Zustände mehrfach besucht werden. Es ist ein sehr wichtiger und häufig verwendeter Algorithmus für Probleme, die nicht in den Speicher passen.

Bei der normalen iterativen Tiefensuche ist der Cutoff-Wert die Tiefe, die bei jeder Iteration um eins erhöht wird. Bei IDA* sind die f-Kosten ($g + h$) der Cutoff-Wert; bei jeder Iteration ist der Cutoff-Wert der kleinste f-Kostenwert eines Knotens, der den Cutoff-Wert bei der vorherigen Iteration überschritten hat. Mit anderen Worten, jede Iteration sucht erschöpfend eine f-Kontur, findet einen Knoten direkt hinter dieser Kontur und verwendet die f-Kosten dieses Knotens als nächste Kontur. Bei Problemen wie dem 8-Puzzle, bei dem der f-Kostenwert jedes Pfads eine ganze Zahl ist, funktioniert dies sehr gut und führt bei jeder Iteration zu einem stetigen Fortschritt in Richtung des Ziels. Hat die optimale Lösung die Kosten C^*, so kann es nicht mehr als C^* Iterationen geben (zum Beispiel nicht mehr als 31 Iterationen bei den härtesten 8-Puzzle-Problemen). Aber für ein Problem, bei dem jeder Knoten unterschiedliche f-Kosten hat, könnte jede neue Kontur nur einen neuen Knoten enthalten und die Anzahl der Iterationen könnte so groß wie die Anzahl der Zustände sein.

Die **rekursive Bestensuche** (*Recursive Best-First Search*, RBFS, ► Abbildung 3.22) versucht, die Funktionsweise der standardmäßigen Bestensuche zu imitieren, benötigt aber nur linearen Platz. RBFS ähnelt einer rekursiven Tiefensuche, doch anstatt den aktuellen Pfad unend-

```
function RECURSIVE-BEST-FIRST-SEARCH(problem) returns eine Lösung oder failure
    solution, fvalue ← RBFS(problem, NODE(problem.INITIAL), ∞)
  return solution

function RBFS(problem, node, f_limit) returns eine Lösung oder failure und eine neue
                                                        f-Kostengrenze
  if problem.IS-GOAL(node.STATE) then return node
  successors ← LIST(EXPAND(node))
  if successors ist leer then return failure, ∞
  for each s in successors do        // f mit Wert aus voriger Suche aktualisieren
      s.f ← max(s.PATH-COST + h(s), node.f)
  while true do
      best ← der Knoten in successors mit kleinstem f-Wert
      if best.f > f_limit then return failure, best.f
      alternative ← der zweitkleinste f-Wert aus successors
      result, best.f ← RBFS(problem, best, min(f_limit, alternative))
      if result ≠ failure then return result, best.f
```

Abbildung 3.22: Der Algorithmus für die rekursive Bestensuche

lich nach unten fortzusetzen, verwendet RBFS die *f_limit*-Variable, um den *f*-Wert des besten *alternativen* Pfads zu verfolgen, der von jedem Vorgänger des aktuellen Knotens aus verfügbar ist. Wenn der aktuelle Knoten diesen Grenzwert überschreitet, wird die Rekursion aufgelöst und die Suche geht zum alternativen Pfad zurück. Bei der Rekursionsauflösung ersetzt RBFS den *f*-Wert jedes Knotens entlang des Pfads durch einen **gesicherten Wert** – den besten *f*-Wert seiner Kindknoten. Auf diese Weise merkt sich RBFS den *f*-Wert des besten Blatts im vergessenen Unterbaum und kann daher entscheiden, ob es sich lohnt, den Unterbaum zu einem späteren Zeitpunkt erneut zu expandieren. ► Abbildung 3.23 zeigt, wie RBFS Bukarest erreicht.

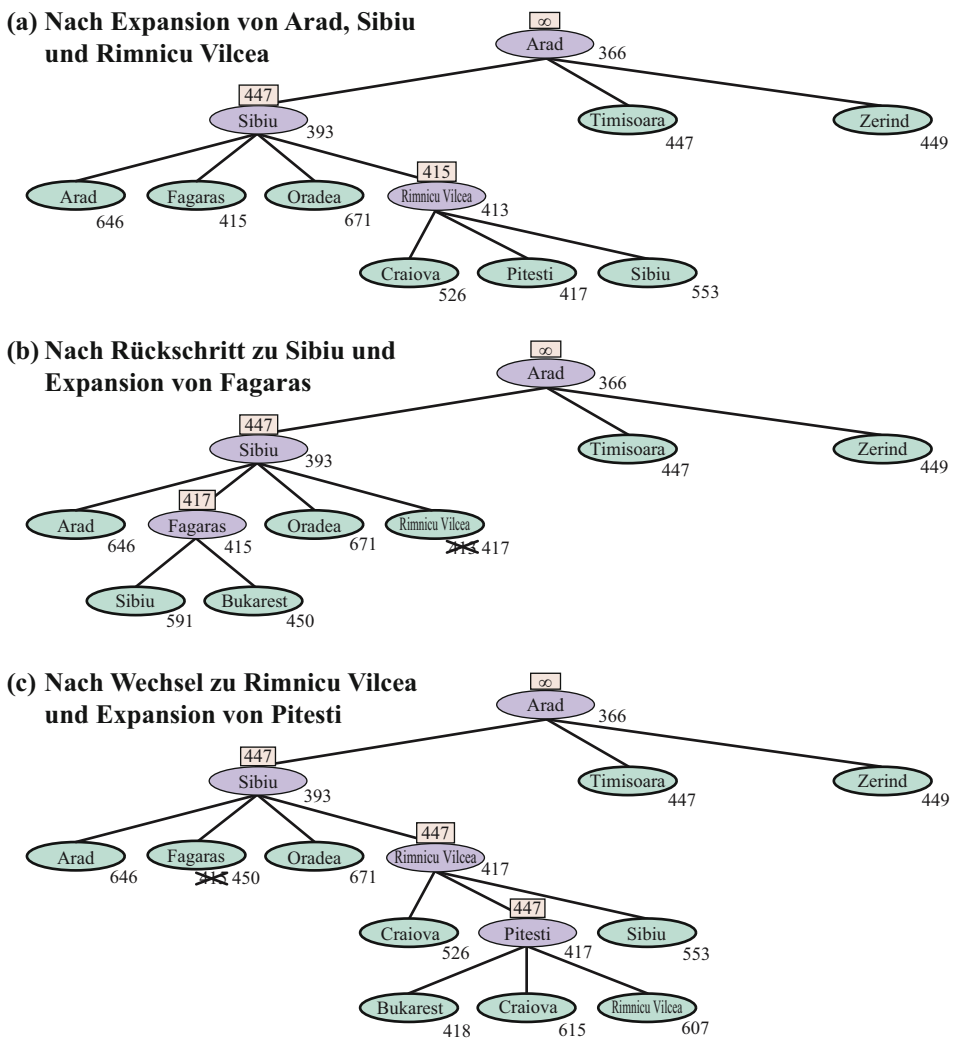


Abbildung 3.23: Phasen in einer RBFS-Suche nach der kürzesten Route nach Bukarest. Der *f-Grenzwert* für jeden rekursiven Aufruf wird über dem jeweils aktuellen Knoten angezeigt und jeder Knoten ist mit seinen *f*-Kosten beschriftet. (a) Der Pfad über Rimnicu Vilcea wird so lange verfolgt, bis das aktuell beste Blatt (Pitesti) einen Wert hat, der schlechter ist als der beste alternative Pfad (Fagaras). (b) Die Rekursion wird aufgelöst und der beste Blattwert des vergessenen Unterbaums (417) wird Rimnicu Vilcea als gesicherter Wert zugeordnet; dann wird Fagaras expandiert, was einen besten Blattwert von 450 ergibt. (c) Die Rekursion wird aufgelöst und der beste Blattwert des vergessenen Unterbaums (450) wird Fagaras als gesicherter Wert zugeordnet; dann wird Rimnicu Vilcea expandiert. Da der beste alternative Pfad (über Timisoara) mindestens 447 kostet, wird die Expansion diesmal bis Bukarest fortgesetzt.