



高级人工智能讲义

AI: 机器对输入响应一个最优的对策或行动

作者：李金龙

组织：中国科学技术大学

时间：11:14, July 14, 2022

版本：0.1.9

用途：研究生《高级人工智能》课程讲义



Can machines think? — Alan Mathison Turing

目录

第一部分

基础知识

第一章 绪论

内容提要

- | | |
|---|---|
| <ul style="list-style-type: none">□ 什么是人工智能□ 图灵测试□ 智能体与人工智能 | <ul style="list-style-type: none">□ 人工智能研究的基本问题□ 人工智能的历史 |
|---|---|

人类发明并制造工具，改造自然，与自然斗争。从算盘到电子管计算机，到微机、个人计算机、手机、互联网、物联网，工具在持续改进和发展。人类制造着越来越便捷和强大的工具，而具备智能的机器成为工具发展的终极目标。工具的强大，突出体现在“智能”二字，随着越来越多的工具冠以“智能”二字，如智能汽车，智能手机，智能家居等等，就逐步引发了人们开始思考“智能”是什么，我们人类如何制造“智能”，如何赋予工具以“智能”。

人工智能随之成为产业界和学术界的热点。

1.1 什么是人工智能

没有人质疑人工智能的重要性，但是当大家坐下来讨论什么是人工智能，如何实现人工智能的时候，就有了不同的看法。

人工智能即“人造的智能”，人工解释为“人造的”，基本没有异议；而智能的定义却引发了众多的争议。有人认为智能是生物体才具备的，具有脑结构才能思考，能思考才能具有智能，具有脑结构的生物体其脑的质量大小被视为智能的指标。而反对者认为非生命体也可以具有智能，甚至抽象的群体或组织也可以具有智能，比如蚁群、鸟群等，只要表现出与智能相关的特征。

思考或思维，是分析、推理和评估等一系列活动构成，与智能形成的过程相关，并不是形成智能的必要条件；没有思考也能展示出智能，比动物对外界刺激的条件反射行为被用来马戏表演，展示动物的数学思维能力和智能。当然，也有人认为这种条件反射并不是真正意义上的智能。

阿兰·图灵提出图灵测试，用黑盒的方式来描述什么是智能，将智能的形成过程视为黑盒，检验从输入到输出的转换能力，并以此来定义智能。

1.1.1 图灵测试

在图灵测试中，有询问者和回答者两个人，他们被物理隔离，看不到彼此，通过计算机进行问答：回答者可能是机器，也可称是真人；机器扮演男性，偶尔撒谎；真人总是诚实回答问题。经过多轮询问与回答，询问者判定回答者是机器还是真人。如果机器成功欺骗了询问者，则机器通过图灵测试，机器被视为具有智能。

通过图灵测试，即意味着机器具有智能。这种采用内部黑盒，仅从外部的输入输出来定义智能的方式，让许多信奉完美主义的人产生质疑。例如内德·布洛克认为，构建一个包含所有图灵测试问题及其回答的数据库，然后通过数据库查询，可让机器通过图灵测试；但他认为这不是机器智能。

而约翰·塞尔也批评图灵测试不能定义真正的智能。举例说，一个人不懂中文，但带着一本中文的规则手册，用中文回答中文问题，能用正确的语法回答正确的内容，能否确认此人懂中文？进一步，

若有一千个不懂中文的人，随机选择一个人借助规则手册用中文回答一个中文问题，那么中文知识存在什么地方？谁懂中文知识？他进一步质询，如果会中文的人回答中文问题，他的中文知识存在什么地方，存在哪一个或哪些个神经元中？

这些批评其实都是希望了解黑盒内部发生的细节过程，追求对输入到输出转换的完美解释。而在现有的科技发展水平，特别是在神经生物学、认知科学等的发展还无法给智能一个公认的定义，没有在智能的形成机制上达成共识。现阶段，我们需要用黑盒的方式来理解人工智能，用计算机处理信息的算法来模拟智能的形成，以解决具体的工程技术问题。

总而言之，人工智能是什么？本书持有的观点：人工智能是抽象的智能体所展示出来的能力。

1.1.2 智能体

现实生活中我们会将一个聪明的机器人视为是人工智能的“化身”，通过其身上的各种高新技术来展示智能。而研究中我们通常会把这种机器人抽象成“智能体（Agent）”，实现摒弃各种不相关的或次要的因素，构造面向具体任务模型的目的。例如，我们定义智能体具有三项基本能力 ODA：观察（Observe）、决策（Decide）和行动（Act），即用 ODA 来抽象机器人的各种能力；我们也可以用类似于 ODA 的说法，用“感知、思维和策略”来抽象机器人的能力。

引入了智能体，我们再将整个世界分成两个部分：智能体和智能体之外一切事物。我们称智能体之外一切事物为机器人身处的“环境”。有了这种对世界的划分，我们就可以定义出智能体。

定义 1.1

agent 能够感知/观察环境，并进行决策和行动的装置。



具体来说，即在给一组外界刺激信号作为输入，智能体“快速地”给出一个“好的”输出。智能体的输入可以是图像、语音、视频或文字等各种数字或模拟的信号，智能体的输出可以是语言、语音、动作或决策等等。智能体的输入和输出的复杂性可以用来描述智能水平的高低。如对输入和输出不加任何限制或约束，这样的智能体是强人工智能的目标；若限制智能体的输入和输出的取值范围，这就是我们现在研究的弱人工智能。人工智能水平的高低，也和“快速的”时间开销和输出的“好的”程度相关。速度快的输出，意味着智能体更聪明；智能体输出的语音、动作或决策更好，也表明智能体更聪明；因此，当我们把智能体视为黑盒，来说明什么是智能，智能水平的高低的时候，我们需要比较智能体输入、输出、响应时间、输出质量等因素，而注意到响应时间主要受到输入输出副主的影响。

1.1.3 人工智能研究的基本问题

“认识自然，改造自然”是科学技术发展的宏伟目标，是人类智力活动的目标和结果。人工智能就是制造聪明的机器替代人类来达成该目标。将自然扩展到人类社会，我们考虑让人工智能来认识世界，改造世界。为达成该目标，人工智能应具备的基本功能包括：

- 描述世界的方法：在智能体或计算机内部描述世界或环境；
- 探索问题求解的通用搜索方法：智能体解决实际问题需要更快的搜索算法；
- 研究智能体的学习能力：智能体如何从经验中学习，提升自身的能力。

1.2 人工智能的起源

1950 年，图灵写了论文“计算机器与智能”一文，提出“机器能否思考”这个问题。当时的计算机在人们眼里就是如同算盘一样的辅助数学计算的器械，而图灵却开始将其和智能关联起来，猜想计算机能否像人一样进行思维。为此，他设计了一种实验方法，即图灵测试，来判定机器是否具有智能。1956 年，约翰·麦卡锡、马文·明斯基和克劳德·香农等人发起了达特茅斯会议，探讨如何构造会思考的机器。达特茅斯会议首次提出了“Artificial Intelligence/人工智能”一词，并明确指出，人工智能研究的目的是制造机器来模仿学习或智能的各个特性，让机器能够懂得语言，可以抽象思维，解决具体问题，同时机器能自我完善。自此，1956 年的达特茅斯会议被视为人工智能的原点，人工智能成为一个独立学科。

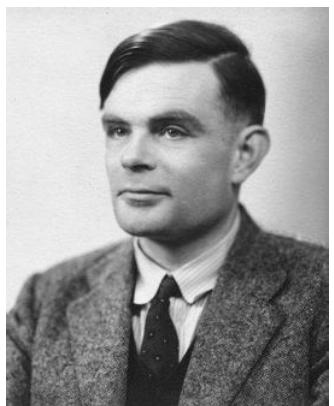


图 1.1: A. Turing

1.3 人工智能的发展历史

人工智能研究的发展离不开人工神经网络，1943 年，美国数学家 Walter Pitts 和神经学家 Warren McCulloch 基于“生理学和脑神经元功能 + 命题逻辑的形式化分析 + 图灵的计算理论”提出了第一个人工神经元模型，人工神经元具有“开”和“关”的状态，能够被激活。1949 年，Donald Hebb，提出 Hebb 学习，来更新网络连边的权值，实现网络学习功能。任何可计算函数都可以用神经元构成的某种网络来实现计算，包括与、或、非等逻辑运算。人工智能研究热潮的起起落落和神经网络研究的进展密切相关，21 世纪初，随着 GPU 的大量使用，深度神经网络流行起来，又带动了人工智能新的一波研究热潮。

我们通过介绍获得图灵奖的人工智能研究学者来了解人工智能的发展历程。

1.3.1 Marvin Minsky

马文·明斯基 (Marvin Lee Minsky)，人工智能之父、计算机科学家。1956 年，和麦卡锡 (J. McCarthy) 等一起发起“达特茅斯会议”并提出人工智能 (Artificial Intelligence) 概念的计算机科学家马文·明斯基 (Marvin Lee Minsky) 被授予了 1969 年度图灵奖，是第一位获此殊荣的人工智能学者。

1951 年，做出世界上第一个神经网络模拟器 SNARE，是最早提出 Agent 概念的人之一。1958 年在麻省理工学院 (MIT) 创建了全球第一个人工智能实验室，明斯基的代表作包括《情感机器》《心智社会》等。1975 年他首创框架理论 (frame theory)，成为通用的知识表示方法。



图 1.2: 明斯基

1.3.2 John McCarthy

约翰·麦卡锡 (John McCarthy)，人工智能之父、计算机科学家、认知科学家。1956 年达特茅斯会议约翰·麦卡锡 (John McCarthy) 提出 artificial intelligence 一词的四人之一，将数学逻辑应用到了人工智能的早期形成中。1971 年获得计算机界的最高奖项图灵奖。

1958 年发明了 LISP 语言，LISP 语言是人工智能界第一个最广泛流行的语言。LISP 是一种函数式的表处理语言。1958 年在麻省理工学院 (MIT) 协助明斯基建立人工智能实验室，1962 年在斯坦福大学建立人工智能实验室。发明了搜索算法中著名的 $\alpha - \beta$ 剪枝算法。

1.3.3 Allen Newell 和 Herbert A. Simon

赫伯特·西蒙 (Herbert A. Simon), 政治学博士、经济学家、计算机科学家, 20世纪科学界的一位奇特的通才, 在多个领域做出了创造性贡献。1978年获得诺贝尔经济学奖, 1975年与学生艾伦·纽厄尔 (Allen Newell) 获得计算机界的最高奖项图灵奖。

1956年, 赫伯特·西蒙参加了达特茅斯会议, 其“逻辑理论家”是当时唯一可以工作的人工智能软件。1957年西蒙与人合作开发了 IPL 语言; 1978年, “经济组织内的决策过程进行的开创性的研究”获诺贝尔经济学奖。1972年7月作为美国计算机科学家代表团成员之一第一次到中国访问, 之后又9次来华访问。1975年和学生艾伦·纽厄尔因其在人工智能、人类心里识别和列表处理等方面进行的基础研究获得图灵奖。1976年提出了“物理符号系统假说”, 成为符号主义学派的创始人和代表人物。1976到1983年之间证明了科学发现只是一种特殊类型的问题求解, 因此也可以用计算机程序实现。在心理学方面, 1960年, 设计实验证明人类解决问题的过程是一个搜索过程, 效率取决于启发式函数, 开发了开发了“通用问题求解系统”GPS。自然语言处理方面, 1970年发展和完善了语义网络, 成为知识表示的通用方法。科学发现只是一种一种特殊类型的问题求解, 因此也可以用计算机程序实现。



图 1.4: 西蒙

1.3.4 Richard Karp

理查德·卡普 (Richard Karp), 计算机科学家。1985年获得计算机界的最高奖项图灵奖。

理查德·卡普研究了与实际应用有密切联系的一系列数学问题, 如路径问题、背包问题、覆盖问题、匹配问题、分区问题、调度问题等, 这些问题都存在“组合爆炸”现象, 提出了经典的“分枝限界法”(branch-and-bound method)。其论文“组合问题中的可归约性”证明了从组合优化中引出的大多数经典问题, 包括背包问题、覆盖问题、匹配问题、分区问题、路径问题、调度问题等, 都是 NP 完全问题。(从某种意义上讲, 人工智能就是研究求解 NP 难题的快速近似算法。)



图 1.5: 卡普

1.3.5 Edward Feigenbaum

爱德华·费根鲍姆 (Edward Feigenbaum), 计算机科学家。1994年获得计算机界的最高奖项图灵奖。

在人工智能初创的第一个10年中, 人们着重的是问题求解和推理的过程。爱德华·费根鲍姆证明了实现智能行为的主要手段在于知识, 在多数实际情况下是特定领域的知识, 从而最早倡导了“知识工程”(Knowledge engineering), 并使知识工程成为人工智能领域中取得实际成果最丰富、影响也最大的一个分支。1965年, 开发出了世界上第一个专家系统程序 DENDRAL。费根鲍姆有句名言: “知识中蕴藏着力量”(In the Knowledge lies the power)。

1963年他主编了《计算机与思想》(Computers and Thought, McGraw Hill), 这本书被认为是世界上第一本有关人工智能的经典性专著。20世纪80年代, 费根鲍姆和 Avron Barr 等人合编了四卷本的《人工智能手册》(The Handbook of Artificial



Intelligence)，内容涵盖了人工智能的理论与实践的方方面面。

1.3.6 Raj Reddy

拉吉·瑞迪（Raj Reddy），计算机科学家。1971年图灵奖获得者约翰·麦卡锡的学生，1994年获得计算机界的最高奖项图灵奖。

拉吉·瑞迪主持过许多大型人工智能系统的开发，例如 Navlab 项目，研发出能在道路上行驶并可跨越原野的自动驾驶车辆；LISTEN 项目，研发语音识别系统帮助解决扫盲问题；Dante 项目，研发火山探测机器人。1979 年他担任国际人工智能联合会议主席时，又带头发起成立了美国人工智能协会 AAAI，并于 1987~1989 年任 AAAI 会长。

1.3.7 Judea Pearl

朱迪亚·珀尔（Judea Pearl），计算机科学家。2011 年获得计算机界的最高奖项图灵奖。

朱迪亚·珀尔的论文《Causality:Models,Reasoning, and Inference》，创立了概率和因果性推理的演算模式，改变了人工智能最初基于规则和逻辑的方向，开创了不确定的条件下的信息处理方向，指出智能系统所面临的不确定性是一个核心问题，并且提出概率论算法作为知识获取及表现的有效基础。



图 1.8: 珀尔

1.3.8 Yoshua Bengio, Geoffrey Hinton and Yann LeCun

约舒亚·本希奥（Yoshua Bengio），杰弗里·欣顿（Geoffrey Hinton）和扬·莱坎（Yann Lecun），2019 年获得计算机界的最高奖项图灵奖。



图 1.9: 本希奥

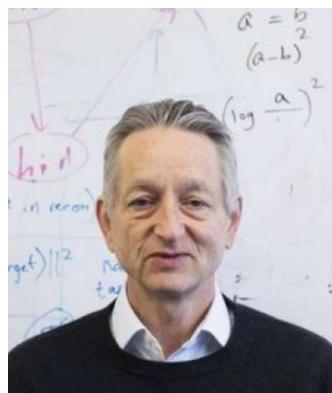


图 1.10: 欣顿



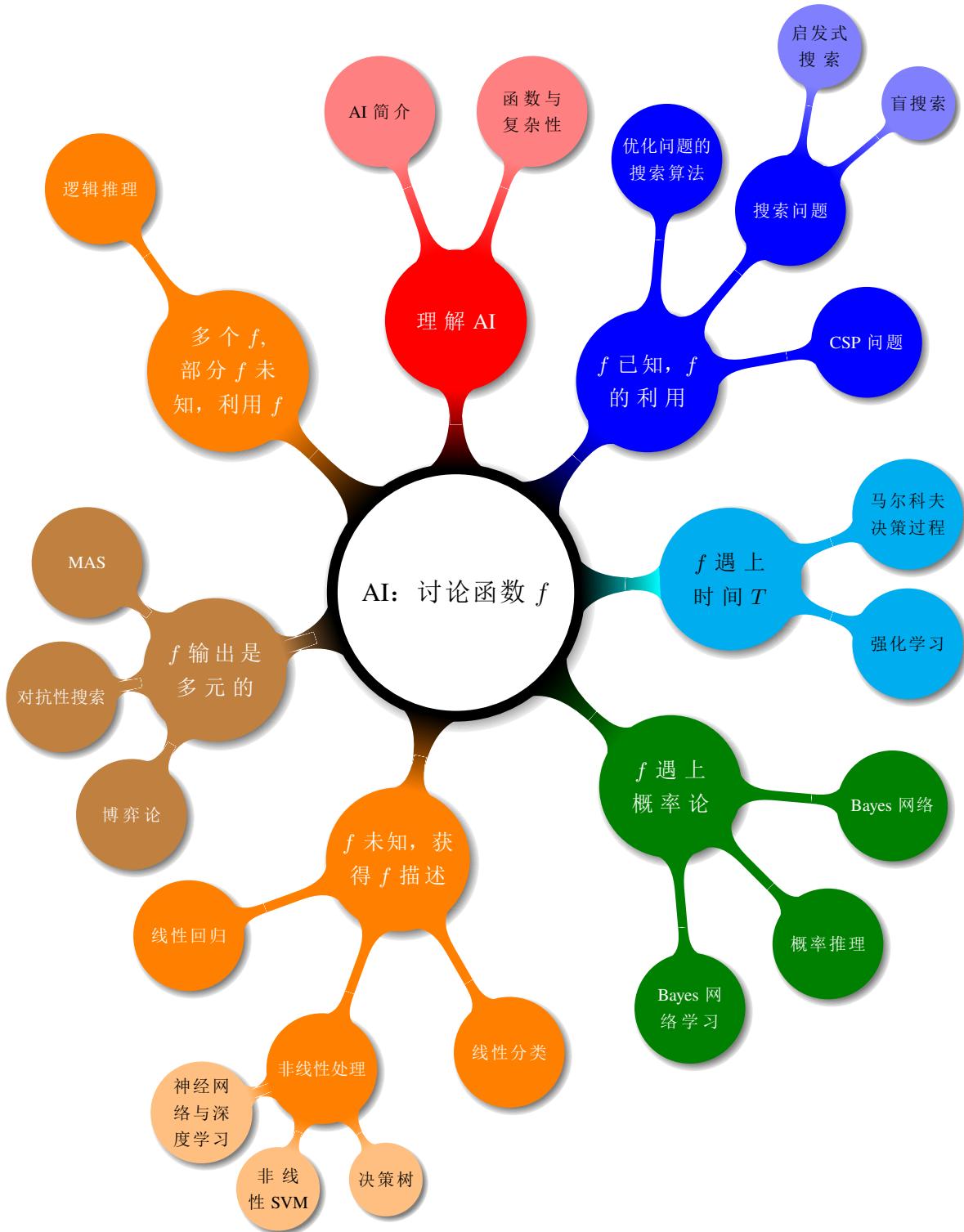
图 1.11: 莱坎

三人从二十世纪末开始推动了深度学习的发展，因他们在深度神经网络领域的研究而获奖。约舒亚·本希奥在自然语言处理领域研究被广泛引用。杰弗里·欣顿被称为“神经网络之父”。扬·莱坎被称为“卷积网络之父”。约舒亚·本希奥和扬·莱坎于 2013 年共同创办了机器学习的顶级国际会议“学习表示国际会议 (ICLR)”。

1.4 人工智能的未来

数据采集、存储和共享因计算机、通信网络和物联网的发展而变得触手可及，由人工智能来自动从数据中发现价值和财富成为未来科技发展的重要内容，影响社会经济生活各个方面。

1.5 人工智能学习路线图



大部分文字材料都来源于互联网，因篇幅问题无法一一指出出处，在此感谢各位原创作者！

第一章 习题

1. 什么是“思维”？机器会思维吗？一个会思维/思考的计算机程序有什么样的特点？
2. “智能”或“智力”具有哪些特点？
3. 你认为具有“智能”的计算机模仿人类解决实际应用问题，应具备哪些基本的能力？每种能力如何界定？
4. 若你来设计一辆无人驾驶汽车，用于在人类城市中行驶，你认为应该包含哪些关键模块，每个模块对应于人工智能的哪些基本能力？
5. 计算机采用“穷举法”解决问题时，逐个尝试问题的解，这是最简单和最直接的基本方法，这和“智能”有什么关联吗？
6. 举两个人工智能的例子，分别阐述黑盒方式和白盒方式制造人工智能的异同。
7. 采用白盒方式制造人工智能，你认为需要具备哪些先决条件？
8. 分别采用“白盒”和“黑盒”实现内部功能的人工智能，应用时最直接差异是什么？

第二章 定义在函数之上的智能体

内容提要

- | | |
|--|--|
| <ul style="list-style-type: none"><input type="checkbox"/> 将智能体抽象成函数<input type="checkbox"/> 函数的定义与描述<input type="checkbox"/> 评价函数描述 | <ul style="list-style-type: none"><input type="checkbox"/> 函数与知识<input type="checkbox"/> 数据是函数的一部分<input type="checkbox"/> 操作给定描述的函数 |
|--|--|

2.1 将智能体抽象成函数

智能体（Agent）是人工智能中的核心概念，是能够感知（Observe）环境，并进行决策（Decide）和行动（Act）的装置。智能体通常被视为人工智能更学术化的同义词，人工智能就是智能体。人们通常用“感知、决策和行动”三个基本功能对各种具体的人工智能装置进行抽象得到智能体，比如将足球机器人，聊天机器人，自动驾驶汽车，棋手“阿尔法狗”等抽象成不同类型的智能体。我们利用函数来归纳总结各种智能体的共性，以函数作为进一步学习人工智能的出发点。

例 2.1（足球机器人）假设有一个足球机器人，能够观察环境，判定形势做出决策，并执行踢足球的各种动作。这个足球机器人的基本功能可以归结为“感知、决策和行动”。它可以感知自身、队友、对手和球的位置；利用这些位置信息，计算或评价各种踢球动作的效果并做出踢球决策；最后将踢球动作付诸行动。

我们将智能体的三个功能视为由智能体的三个关键组成部件：感知器、决策器和执行器来完成。层出不穷的各类感知器扩展了人类的感觉器官，使得智能体感知能力提升，完成后续决策和执行的潜力超过人类。

例 2.2（足球机器人的感知器）足球机器人的感知器如同人的眼睛和耳朵，可看可听；也可以通过视觉或电磁波感知位置，通过风向风速传感器感知风，通过记录和计算感知球的速度以及其它机器人的残余能量等等，感知器的输出就是决策器的输入。

感知器完成光、电、声和电磁等各类原始信号的采集和转换，采集原始信号的种类和范围直接影响智能体的完成任务的潜力。

例 2.3（足球机器人的执行器）足球机器人的执行器如同人的手、脚和头，可抓可踢可顶；足球机器人执行器接收决策器的输出为输入，然后从事先确定的、它能完的有限个动作集合中选择出对应的动作，再执行该动作规定的操作指令。踢球的力量、方向等是踢球动作的连续型参数，在基于计算机的智能体中被离散化为多个，但有限的动作。

智能体的感知器和执行器在完成具体的实际任务时，和硬件密切相关；在投入使用时，其设计和制造需要考虑各类成本、功能和性能的均衡。我们的课本将略过感知器和执行器的设计和制造，将其简化为决策器的输入和输出。感知器和执行器可以视为功能更专业、更简单的智能体。

例 2.4（足球机器人的决策器）足球机器人的决策器如同人的大脑，用于思考和决策。足球机器人的决策器将感知器采集到的环境信息做为输入，经过一个确定性的过程/操作指令序列，转换为踢球动作，并输出给执行器。其中的操作指令序列模仿了人类大脑的思维过程。

站在计算机程序设计的角度来看，智能体的决策器就是计算机算法，将已知条件/输入转换为结论/输出。将输入转换为输出，还有一种更常见的说法，就是“函数”。

2.2 函数及其描述

实际生产生活中，我们会把从输入到输出的转换视为一种“函数”或者“映射”。例如，函数可以是生产过程，把原材料转换成产品；函数可以是思维过程，把中文翻译成英文等等。现实中的各种转换，都可以视为某种函数。

2.2.1 函数的定义

在数学领域，函数至少有两种定义严谨的定义：定义2.1和定义2.2。

定义 2.1 (函数 1)

一般的，在一个变化过程中，有两个变量 x, y ，如果给定一个 x 值，相应的就确定唯一的一个 y ，那么就称 y 是 x 的函数，其中 x 是自变量， y 是因变量， x 的取值范围叫做这个函数的定义域，相应 y 的取值范围叫做函数的值域。



定义 2.2 (函数 2)

设 A, B 是非空的集合，如果按照某种确定的对应关系 f ，使对于集合 A 中的任意一个元素 x ，在集合 B 中都有唯一确定的元素 y 和它对应，那么就称 f 为从集合 A 到集合 B 的一个函数，记作 $y = f(x), x \in A$ 或 $f(A) = \{y | f(x) = y, y \in B\}$ 。



我们可以粗略地理解函数为“两个集合 A, B 的元素之间的关系”。这种关系或转换，即，从 x 到 y 的关系/转换，它的一种理解和使用方式就是“已知 x 的值，可由函数得到 y 的值”。

2.2.2 函数的描述

函数的定义是给出函数的功能描述，而函数的描述是指把函数在载体上表达出来，例如在纸上把函数书写出来。函数的描述涵盖了函数的定义，并额外要求用某种载体把函数展示出来。本书是指在计算机内部把函数表达出来，更确切地说是在计算机内存中把函数表达出来。

函数的描述要考虑描述的精确性、简洁性和使用方便性。例如，把函数 $y = ax + b$ 在记事本里写成一个字符串，这种描述简洁，精确但是使用不方便；用 C 语言把该函数的计算过程实现，这种描述简洁、精确，也易用。

知识点 2.1 (语言叙述法)

使用语言文字来描述函数的关系。



例如“天下雨了，地上变潮湿了”，“打开开关，灯亮了”等。这种表示方法在现实中是最常见的，描述了生活中的各种变换现象。通用性是语言叙述法的最大优点，处处可用；但是精确性不足是语言叙述法最大的问题，虽然处处可用，但是效果和性能难以保证工程效果，所以工程上基本不用。

知识点 2.2 (图像法)

把一个函数的自变量 x 与对应的因变量 y 的值分别作为点的横坐标和纵坐标，在直角坐标系内描出它的对应点，所有这些点组成的图形叫做该函数的图象。



图象法描述函数的优点是通过函数图象可以直观、形象地把函数关系表示出来；缺点是从图象观察得到的数量关系是近似的，不够精确，因而就限制了其工程应用范围。

知识点 2.3 (列表法)

制作一个两列的表格，将定义域的元素排在表的左边列，右边列给出元素在值域对应的值。



任何定义在有限、可列定义域上的函数理论上都可以用列表法精确描述。面对有连续型定义域或定义域包含无穷个元素的函数时，计算机离散化处理可以使该条件得到满足，但列表法依然只是理论上能精确描述各种函数，工程上可能无效。因为用列表法表示一个函数时，若一行一个定义域的元素，可能面临表中的行太多的问题。所谓行太多，是指工程上，表中过多的行无法在计算机存储设备中完整保存下来。

知识点 2.4 (解析法)

用数学等式关系来描述函数。



例如“ $y = \sin(x)$ ”，“ $\mathbf{y} = A\mathbf{x} + \mathbf{b}$ ”等。精确、简洁和易使用是解析法的优点。能用解析式简单描述出来的函数，表示两个集合（定义域和值域）元素之间有很强的关系，但是实际生产生活中，解析法能描述的的变换非常有限，复杂的变换或函数无法或很难用解析法来表示。注本书中数学公式中的黑体字表示集合相关的概念，如集合、向量、矩阵；白体字表示标量

各种函数描述方法中，解析法因其精确、简洁和易使用，成为我们追求的目标。如何弥补解析法描述复杂函数的不足，扩展解析法的应用场景，使之能表示任意函数成为我们考虑的重点。本书考虑以计算机为载体的函数描述方法，并结合人工智能的工程实践展开讨论。

2.2.3 计算机中的函数描述

在计算机内存中描述一个函数，有多种可行的方法，上述的列表法，解析法，图像法和语言叙述法都可以使用，但在精确性、简洁性和易用性方面各有优缺点。例如用图像描述一个函数，除了让我们人类用视觉观察图像特征，其他应用都很困难；用一段文字（语言叙述法）描述一个函数，在现有的自然语言处理水平下，我们几乎无法用这个函数完成任何有用的任务。

知识点 2.5 (在计算机中用列表表示函数)

给定函数 $y = f(x)$ ，其中 $x \in \mathbb{X} = \{x_1, x_2, \dots, x_l\}$ ， $l = |\mathbb{X}|$ 表示定义域的大小，可如表格2.1所示，通过穷举定义域来描述该函数。

x	$y = f(x)$
x_1	$y_1 = f(x_1)$
x_2	$y_2 = f(x_2)$
...	...
x_l	$y_l = f(x_l)$

表 2.1：用表格定义函数的所有对应关系， $l = |\mathbb{X}|$



知识点 2.6 (在计算机中用列表表示多变量函数)

给定函数 $y = f(\mathbf{x})$, 其中输入 $\mathbf{x} = (x_1, x_2, \dots, x_n) \in \mathbf{X} = (\mathbf{X}_1, \mathbf{X}_2, \dots, \mathbf{X}_n)$ 是一个向量, 输入变量的第 i 个分量的值域大小记为 $|\mathbf{X}_i|$, 其第 j 个取值为 v_{ij} , $l = |\mathbf{X}|$ 表示定义域的大小, 可如表格2.2所示, 通过穷举定义域来描述该函数。

x_1	x_2	...	x_n	$f(x_1, x_2, \dots, x_n)$
v_{11}	v_{21}	...	v_{n1}	y_1
v_{11}	v_{21}	...	v_{n2}	y_2
...
$v_{1 \mathbf{X}_1 }$	$v_{2 \mathbf{X}_2 }$...	$v_{n \mathbf{X}_n }$	$y_{ \mathbf{X}_1 \mathbf{X}_2 \dots \mathbf{X}_n }$

表 2.2: n 元函数的表示

在计算机科学中用列表法表示函数, 简单直观, 当算法是对表格的操作时, 便于对算法的时空复杂性进行分析, 这一思想将贯穿我们整本书。

知识点 2.7 (在计算机中用一段程序代码描述函数)

程序代码段通常用程序设计语言表达的函数解析式。例如 C 语言中的数学库的正弦函数 `double sin(double x)`。



具有解析表达式的函数通常可以用程序代码段实现其功能; 而程序代码段也可以实现无解析表达式的函数, 例如列表表示的函数。回归人工智能这一主题, 设计和制造智能体, 就是要写或生成一段程序代码/算法, 完成从输入到输出的转换。如何构造这段程序代码, 高效地完成输入/输出转换任务, 成为人工智能研究的核心和热点问题。

2.3 评价函数描述

我们说函数描述要求精确、简洁和易用, 在工程应用中我们有些更实用的评价指标: 完备性、准确性和复杂性。

2.3.1 函数描述的完备性和准确性

知识点 2.8 (完备性)

函数描述对任意的合法输入都能返回一个输出。



若有一个合法输入, 函数描述无法给出输出, 那么这个函数描述就是不完备的。

知识点 2.9 (准确性)

函数描述对任意的合法输入, 返回的输出都是最佳应对。



如果函数输出的不是对应输入的最佳应对, 那么就存在误差, 所有定义域上输入得到输出的误差之和可用于度量函数描述的准确性。

2.3.2 函数描述的复杂性

人工智能的智能体被抽象成了函数，而函数对应于计算机程序代码/算法，计算机算法的时空复杂性分析是计算机科学研究永恒的主题，是程序设计考虑的重要因素。很多工程领域的学者认为人工智能就是设计和实现算法来解决高度复杂的问题；或者说是针对 NP 完全问题设计和实现有效的近似算法。而函数描述的复杂性关乎算法的时空复杂性，因此，在人工智能研究中，我们必须要考虑函数描述的复杂性，甚至将函数描述的复杂性当作我们分析和讨论的焦点。

定义 2.3 (函数描述的复杂性)

计算机中一个函数占有内存的大小。



定义2.3清晰和简单，但是在实际工程应用中，函数描述载入内存中时，其占有的内存大小还受到很多其它因素的影响，比如程序设计语言，编译系统，操作系统等。因此，有多种用于分析和讨论的函数描述复杂性的变种。

知识点 2.10 (用列表法的行数表示函数描述的复杂性)

用列表法表示函数时，一行表示定义域的一个值，简化考虑，假设列表的一行占有一个单位的内存空间。此时函数描述的复杂性就是函数定义域的大小。例如表2.2所示的 n 元多变量函数 $y = f(\mathbf{x})$ ，其描述复杂性为 $|\mathbf{X}_1||\mathbf{X}_2| \dots |\mathbf{X}_n| \geq 2^n$ ，表示描述函数需要占有的内存空间超过输入变量个数 n 的指数函数 2^n ，说明当 n 较大时，列表法描述函数，复杂性将导致工程上无法实现。



这种穷举“定义域-值”的列表法实现的函数描述具有最大复杂性。与之相对的，我们可以定义函数描述的最小复杂性——最小描述长度。

知识点 2.11 (最小描述长度原则)

在一个函数的所有描述构成的有限集合中，能产生最大压缩效果的那个描述是最小的，也是最好的。



最小描述长度在 1978 年由 Jorma Rissanen 引入。定义最小描述长度原则背后的基本想法是：将列表形式的函数描述视为一个资料集合或数据集，而此资料集内的任一规律性都可用来压缩资料。也就是说在描述资料时，与逐字逐句来描述资料的方式相比，能使用比所需还少的符号是我们设计设计或选择函数描述的基本原则。最小描述长度定义简单，实现困难。我们希望选取到的描述能抓到资料中最多的规律，于是我们则寻找压缩效果最佳的描述。

最小描述长度原则是对奥卡姆剃刀原理的形式化后的一种结果。奥卡姆剃刀 (Ockham's Razor) 原理由 14 世纪逻辑学家、圣方济各会修士奥卡姆的威廉提出，“如无必要，勿增实体”；许多近现代的科学家也提出了类似的思想，如牛顿提出的“如果某一原因既真又足以解释自然事物的特性，则我们不应当接受比这更多的原因”原则，爱因斯坦的“万事万物应该都尽可能简洁，但不能过于简单”等。

人们对事物描述的复杂性，从朦胧的感性认识逐步上升到理性，上升到形式化，数量化度量。我们注意到任一函数/列表都可以由一有限（譬如说，二进制的）字母集内符号所成的字串来表示，所以可通过精确量化一个字符串 s 的描述来进行讨论。

知识点 2.12 (柯尔莫哥洛夫复杂性)

所有能输出字符串 s 的程序中，最短程序的长度。



柯尔莫哥洛夫复杂性描述程序的复杂性，也称为算法熵。函数的列表描述可以看成是字符串 s ；资料集/数据集可以看成是字符串 s ；程序代码可以看成是字符串 s ；一段语言可以看成是字符串 s 。字符串 s 可以找到内在规律后，用一个较短的程序压缩，执行该程序就会输出字符串 s ，因此该程序就等价于字符串 s 。柯尔莫哥洛夫复杂性衡量了描述一个（字符串）对象 s 所需要的信息量；不同程序设计语言，同一功能的程序代码长度会不一样，一般给定程序设计语言来进行柯尔莫哥洛夫复杂性的讨论。评价函数的描述长度可以采用前面提到的各种方法，但在人工智能中用得较多的是列表法描述和解析表示法。列表法具有最大描述长度，用作评价其它函数描述的基准；解析表示法用数学等式关系中参数的个数来表示描述长度。

知识点 2.13 (解析式的复杂性)

函数解析式的参数数量。



例如，机器学习中的线性表达式 $y = Ax$ ，其描述长度为 $l = “A”$ 的分量个数”。

2.4 函数与知识

我们认为世界上事物之间的转换就是函数，函数又是智能体的抽象，制造利用这种表示转换的函数，就是制造利用智能体。这种事物间的转换，或者说转换规则和过程，用计算机程序设计语言写出来就是一种函数描述。函数描述体现的是事物之间内在的必然联系，即“规律”，而规律是“知识”的一种不同表述。因此我们认为函数就是知识，描述事物之间的转换或联系。

关于“知识是什么”现在还没有统一的认知和定义。哲学上的争论，包括柏拉图的“确证的真信念 (JTB)”，Edmund Gettier 的两个反例等，对我们工程上定义和利用知识并没有太多的帮助。在人工智能领域，关注知识并首次提出“知识工程”的爱德华·费根鲍姆认为人工智能的关键在于知识。现今流行的机器学习，其目标是寻找表示事物转换的函数描述，它有一个同义的名称“知识发现”。

2.5 数据是函数描述的一部分

当我们把感知器采集到的数据或智能体的输入存入列表/表格中，并指定表格中的部分列为函数的输入变量 x ，部分列为函数的输出变量 y ，那么这个表格及其中的数据就是列表法函数描述的一部分。

知识点 2.14 (列表法的部分函数描述)

如表2.3所示，其中 $x \in \mathbb{X} = \{x_1, x_2, \dots, x_l\}$ ， $l = |\mathbb{X}|$ 表示定义域的大小， $0 < k < l$ 是数据集的行数。

x	$y = f(x)$
x_{i1}	$y_{i1} = f(x_{i1})$
x_{i2}	$y_{i2} = f(x_{i2})$
...	...
x_{ik}	$y_{ik} = f(x_{ik})$

表 2.3: 用表格存储数据集, k 是数据集大小

表2.1完整描述了函数 $y = f(x)$, 表2.3是表2.1的一个真子集, 描述了函数 $y = f(x)$ 的部分内容。

如何从列表法的部分函数描述中得到解析形式的完整函数描述, 是人工智能研究的第一个基本问题——学习问题, 也叫做机器学习。

从函数描述的评价准则出发, 机器学习要完成两个强制(必须做到的)任务和一个优化任务:

- 完备性: 补充函数描述, 对定义域内的每个值 x 都找到一个应对 y ; 属强制任务;
- 复杂性: 选择或设计一个简洁的解析式, 用于压缩描述长度; 属强制任务;
- 准确性: 尽可能让定义域上的 x 都对应到最佳的 y 值, 错误的应对(误差)越小越好; 属于优化任务。

当受到工程应用条件的限制, 两个强制任务很难完成时, 可以把强制任务改成可选任务, 这时机器学习就可以被称为数据挖掘。

2.6 操作给定描述的函数

人工智能研究的第二个基本问题是函数的描述给定时, 如何操作该函数, 即函数的使用。假设函数已经以某种描述形式给出, 我们讨论不同描述形式下函数的几个典型操作及其时空复杂度。这些操作包括:

- 操作一: 已知 x 的值, 求对应 y 的值;
- 操作二: 当一个 y 值满足特定条件时, 求对应 x 的值;
- 操作三: 当一组 y 值满足特定条件时, 求对应 x 的值;

2.6.1 给定列表法的函数描述

列表法给出的函数描述, 行数不可能太多, 否则内存放不下。假设函数描述有 l 行, 空间复杂度为 $O(l)$ 。注对于 n 元函数, 其列表法描述的行数/空间复杂度达到 $O(l = 2^n)$ 。

例 2.5 (操作一的时间开销) 给定列表法的函数描述时, 若比较 1 次花费 1 个单位时间, 那么操作一的平均时间复杂度为 $O(l)$ 。

$O(l)$ 的时间复杂度属于穷举法。若可以排序、构建索引或构建哈希表, 操作一的计算时间开销可降低, 甚至降到对数量级 $O(\log l)$ 。

例 2.6 (操作二的时间开销) 给定列表法的函数描述时, 若比较 1 次花费 1 个单位时间, 那么操作二的平均时间复杂度不会优于 $O(l)$ 。

典型的关于 y 的单个条件有: 特定的 y 值, y 的最大值, y 的最小值, y 的某个极值等。基于列表法描述的函数, 由 y 值计算 x 的值, 方法基本类似与操作一, 区别在于: 由 x 计算 y , 通常只会返回

一个 y 值；而由 y 计算 x ，可能返回 0, 1 或多个 x 值。

例 2.7 (操作三的时间开销) 给定列表法的函数描述时，操作三的平均时间复杂度与条件中不同 y 的个数 k 相关，通常不低于 l 的 k 次多项式量级 $O(l^k)$ 。

例题 2.1 (*TSP* 问题) 若 $y_{ij} = f(x_i, x_j)$ 表示完全无向图 G 的边 (x_i, x_j) 的权值 (边的长度)，求经过恰好所有顶点一次，并回到起点的最短路径 (最短哈密顿回路)。穷举法解决该问题的时间开销是多少？

解 图表示为边表形式 $\{(x_i, x_j, y_{ij})\}$ ，这就是函数 f 的列表法描述，设表格的行数为 $l = n(n - 1)/2$ ，其中 n 为图 G 中顶点的个数。给定起点时，哈密顿回路的总数有 $(n - 1)!$ 种，采用穷举法完成操作三的精确计算，时间复杂度为 $O((n - 1)!)$ 。

现在提出了很多求解 *TSP* 问题的快速算法，但只能获得近似最短哈密顿回路，无法保证最优。非穷举的、通用的、精确计算最短哈密顿回路的方法至今仍有待研究。

2.6.2 给定解析式的函数描述

一个函数解析式的复杂度在人工智能领域内通常用函数/模型的参数数量来衡量，比如 2021 年微软和英伟达推出的语言模型 MT-NLG 包含 5300 亿个参数。尽管这些函数解析式的参数数量越来越多，但是我们在分析这些解析式，做理论上的讨论时，仍会假设其时间代价是常数量级。

例 2.8 (操作一的时间开销) 给定解析式的函数描述时，操作一的平均时间复杂度为 $O(1)$ 。

输入不同的 x 的值，函数解析式计算的时间开销基本不变，尽管不同的解析式之间计算开销差别会非常巨大。

例 2.9 (操作二的时间开销) 给定解析式的函数描述时，操作二的平均时间复杂度与解析式的具体形式相关，没有统一的结论。

指定 y 值的解析式函数，求 x 就是解方程，但是解方程不存在通用的方法，求解方法依赖于解析式的具体形式。求 y 的最大值或最小值，是优化问题；而大多数关于有解析式函数的优化问题是 NP 完全的，一般不存在能获得最优解的通用的快速解法；目前能实现精确求解的算法是穷举，但是需要花费大量时间开销。

例 2.10 (操作三的时间开销) 给定解析式的函数描述时，操作三的时间开销一般情形下都会高于操作二，也是依赖于具体的函数解析形式。



笔记 当函数以某种形式被描述出来，对函数的操作可以依据描述形式进行设计和实现，而所有的操作有一个通用的实现方式——搜索。函数的操作是问题求解的过程，所以搜索是通用的问题求解方法。

ℳ 第二章 习题 ℳ

1.

第二部分

问题求解

第三章 搜索基础

内容提要

- 从图上的最短路径开始
- 基于搜索的问题建模
- 什么是搜索
- 搜索算法框架

3.1 从图上的最短路径开始

例题 3.1 (图上的最短路径) 给定一个图 $\mathbb{G} = (V, E)$, 设源点为 I , 目标为 G , 求从 I 到 G 的最短路径。

解 分不同的情况进行讨论:

1. 若边无权或边权都相等时, I 到 G 的最短路径就是边数最少的路径。可采用图的广度优先搜索遍历算法, 以 I 为起点, 找到 G 时, 结束算法运行; 采用邻接表实现的算法的时间复杂度为 $O(n+e)$, 其中 n 为图的顶点个数, e 为图中边的数目。
2. 若边上由不完全相同的权值, I 到 G 的最短路径就是路径上边权值和最小的路径。可采用 Dijkstra 算法, 以起点 I 为中心向外层层扩展, 计算节点 I 到其他所有节点的最短路径, 算法时间复杂度 $O(n^2)$; 也可采用 Bellman-Ford 算法, 允许边权值为负数, 算法时间复杂度 $O(ne)$; 或采用 Floyd 算法, 可以计算任意源点到目标的最短距离, 算法时间复杂度 $O(n^3)$, 空间复杂度 $O(n^2)$ 。

思考一下, 当我们变成解决例题3.1时, 顶点是如何表示的? 用一个具有唯一性的“编号”标识一个顶点。计算机程序设计中数据结构设计的基本思路是用图的一个顶点来抽象表示要处理的一个数据对象。在人工智能中, 我们会用顶点(数据对象)的属性向量来表示一个顶点, 而不是一个标量形式的“编号”。

例 3.1 (最短路径的特例) 若例题3.1中顶点用属性向量 (a_1, a_2, \dots, a_m) 表示, 其中 $a_i \in \mathbf{A}_i$ 是第 i 个属性, $i = 1, 2, \dots, m$, m 为顶点属性个数, a_i 的值域 \mathbf{A}_i 的大小 $|\mathbf{A}_i| \geq 2$, 求图 \mathbb{G} 上给定源点 I 和目标 G 的最短路径。

这个例子中我们仅仅将原始最短路径问题的顶点表示, 结合具体的实际工程应用, 用特定的属性向量描述出来。而此时, 我们可以发现图 \mathbb{G} 中可能的顶点总数为 $n = \prod_{i=1}^m |\mathbf{A}_i| \geq 2^m$, 这意味着当 m 较大时, 例如 $m > 100$, 现有的计算机难以存储和计算这种大规模的图。而在大数据时代, 数据对象的属性个数 m 成百上千是非常常见的, 我们还讨论千万、上亿量级属性个数的数据, 因此, 前述求最短路径的算法, 其时间复杂度都高于 $O(n)$, 即不低于 $O(2^m)$ 。

若我们能把求最短路径的算法的时间复杂度从顶点数 n 的线性量级 $O(n)$ 降低到 n 的对数多项式量级 $O(p(\log n))$, 也就是 m 的多项式量级的时间复杂度 $O(p(m))$, 那么这个算法就被称为求最短路径的快速算法。这种快速算法时间复杂度低于 $O(n)$, 属于次线性时间复杂度的算法类, 找最短路径的过程中没有完整地访问过所有顶点, 通常也不能保证一定能找到最短路径, 是一类找最短路径的近似算法。**注** $p(\cdot)$ 是多项式函数。

知识点 3.1(人工智能中求最短路径的算法)

要求算法的时间复杂度为顶点数目 n 的对数多项式量级, 即 $O(p(\log n))$, 其中 $p(\cdot)$ 表示多项式函数。



在人工智能中, 并非总是要求寻找图 G 上源点 I 到目标 G 的最短路径, 有时只要求找到一条存在的路径, 这个任务有时也是非常困难的。我们把这个任务称之为“搜索”。

3.2 什么是搜索

3.2.1 组成搜索的五要素

定义 3.1 (搜索/Search)

如图3.1所示, 给定状态空间 S , 后继函数 $\text{succ} : S \rightarrow 2^S$, 初始状态/初态 s_0 , 目标测试 $\text{GOAL} : S \rightarrow \{\text{True}, \text{False}\}$ 和路径耗散 c_{ij} 五个要素的描述, 找到从初始状态 s_0 出发到使得目标测试 $\text{GOAL}(s)$ 为真的状态 s 的一条路径。

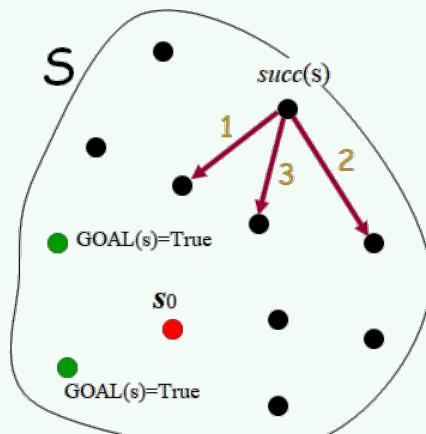


图 3.1: 搜索的五要素



定义3.1给出的搜索是基于“状态”的定义。状态是事物的描述的一种方法, 通常表示成事物属性构成的向量, 比如一个围棋棋盘, 自动驾驶汽车感知到的外部环境, 聊天机器人听到的一句话等等。图3.1中, 一个点代表一个状态, 所有的点构成状态空间 S , 红色的点是初始状态, 绿色的点是满足目标测试为“真/True”的状态, 状态空间中的状态通过后继函数 $\text{succ}(\cdot)$ 在不同状态之间连上了“边”, 由此将状态空间变成了类似于图3.2所示的“状态图”, 边上的权值表示路径耗散 c_{ij} , 可理解为边的长度。图3.2中路径耗散/边权值 c_{ij} 为常数, 故可以略去不标出。

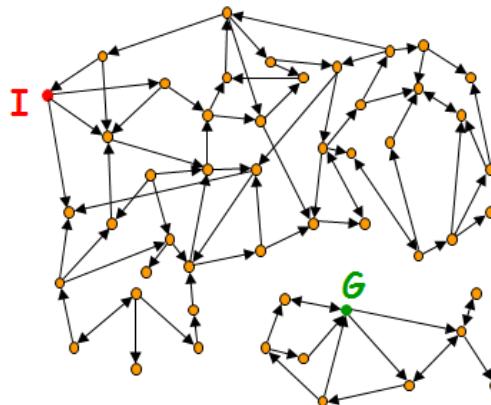


图 3.2: 描述搜索的状态图

注 状态图可以是有向图，也可以是无向图。状态图可以是连通的，也可以不连通。

3.2.2 搜索问题的解

搜索问题是为了解得从初态到使得目标测试为真的目标状态集合的一条路径。通常不要求找到从初态到所有目标状态的路径，找到一条到某个或任一个目标状态的路径即可，例如下棋，目标状态为使得目标测试为“胜利”的状态集合，这类表示“胜利”的状态很多，不需要走到某个特定的“胜利”目标状态，只需要找到其中任意一个即可。

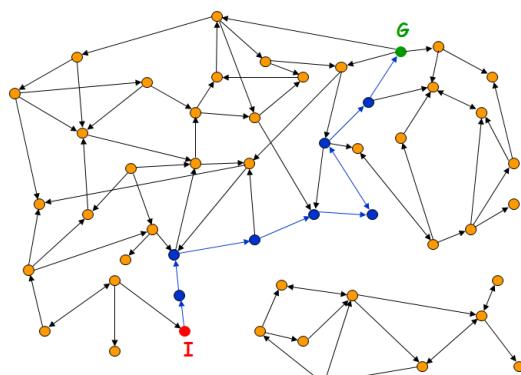


图 3.3: 描述搜索的状态图：初态和目标状态位于同一个连通片

如图3.3所示的状态图，初态 I 和目标状态 G 位于同一个连通片，它们之间一定存在连通的路径，图中用蓝色边标记除了一条从 I 到 G 的路径，作为对应搜索问题的一个解；而图3.2中，初态 I 和目标状态 G 分别位于状态图的两个不同连通片上，故不存在从 I 到 G 的路径，因此对应的搜索问题无解。

知识点 3.2 (连通图上搜索问题的解)

若描述搜索的状态图是连通的，或者搜索中的初态和目标状态位于状态图的同一个连通片，那么搜索问题一定有解，代表解的路径数量大于等于 1。

这个结论很容易用反证法，由状态图的连通性证明。对应的，我们也有关于搜索问题无解的结论。

知识点 3.3 (搜索问题无解的条件)

若搜索问题的任意一个（初态，目标状态）对都不出现在状态图的同一个连通片中，那么该搜索问题无解。

知识点3.3用反证法易证明。

知识点 3.4 (最短路径与最优解)

若搜索问题有有限多个解，那么称其中长度（路径耗散）最短的路径称为搜索问题的“最优解”。

对于边无权的状态图来说，每条边上的路径耗散都是常数 $c > 0$ ，所以最短路径就是边数最小的路径；而对于边权值不完全相同的状态图，最短路径是指路径上所有边的路径耗散的和最小者。

最优解是很多实际问题要求的目标，比如机器人导航寻找最短路径。

例 3.2 (机器人导航) 如图3.4所示地图，有不可触碰的障碍物(咖啡色所示)，可能是游戏地图，可能是汽车自动驾驶地图等；若设机器人只能按照图片中的网格线移动，每个网格点视为图的一个顶点，每条边的长度为1，地图的这种网格化得到一个图，其中红点是出发点，绿点是目标终点；寻找图中一条从红点到绿点的最短路径。

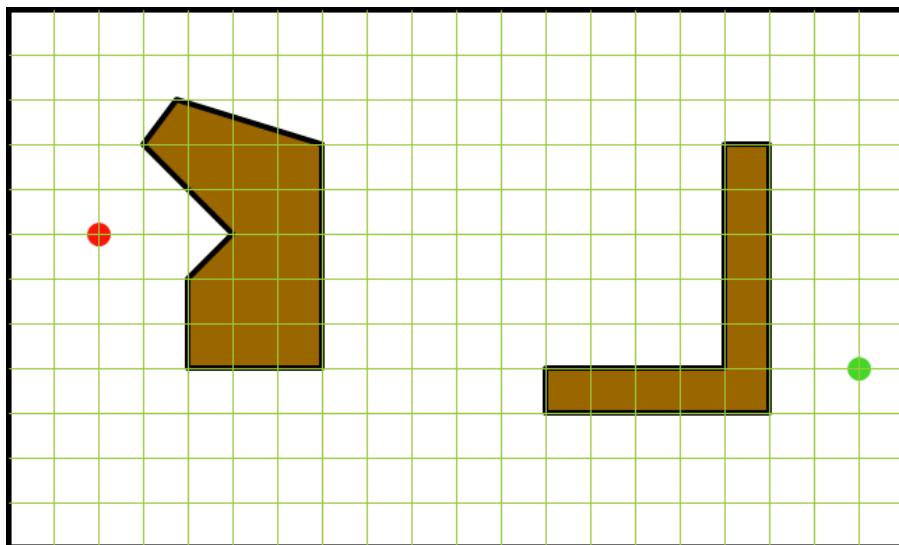


图 3.4: 机器人导航

例 3.3 (数码问题) 给定一个 $n \times n$ 的方格棋盘，将数字 $1 \sim n^2 - 1$ 填入棋盘中，如图3.5所示的 $n = 3$ 的数码问题，其中有一个空白格子，唯一允许的操作是可以将数字移动到相邻且有长度为1的“共边”的空白格子中。从某个给定的初始棋盘，能否找到一个移动序列，得到一个指定的目标棋盘？例如图3.6所示的一个悬赏求解的数码问题。

1	2	3	4
5	10	7	8
9	6	11	12
13	14	15	

图 3.5: 8-数码问题



1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

?

1	2	3	4
5	6	7	8
9	10	11	12
13	15	14	

=\$1000

图 3.6: 悬赏的数码问题

数码问题在 1878 年由 Sam Loyd 引入，他自称为美国最伟大的数码游戏专家，他提出了一个如图3.6所示的问题，当时悬赏 1000 美元求解，一百多年过去了，至今无人领赏。无人领赏的原因是不存在从初始棋盘到目标棋盘的路径。当我们把棋盘建模成搜索问题的状态，若两个棋盘间移动一个数字就彼此可以互达，我们在这两个棋盘状态间连一条边，这样我们得到一个状态图。图3.6所示的搜索问题，初始状态和目标状态在两个状态图的两个不同连通片上。

知识点 3.5 (搜索问题“松弛”的解)

一个搜索问题返回的解是从初态到使目标测试为真的目标状态的一条路径，如果问题只需要返回路径的目标状态，不需要经过的中间状态，那么这个搜索问题对解的要求“降低”了，得到了一个“松弛”的解。



八皇后问题和数独问题是这类解要求降低了的搜索问题的典型代表，这是一类搜索问题的变型。更一般地，搜索问题中的特殊类型——优化问题和约束满足问题都属于这类变型。

3.2.3 状态与状态空间

人工智能的要设计制造智能体解决现实世界的各种具体问题，可以把智能体和智能体之外的世界（环境）视为一个系统中两个独立的部件，系统的两个部件进行交互，交互模式为：智能体/函数接收环境的“状态”为输入，然后完成从一个状态到另一个状态的转换，输出环境的另一个“状态”。其中状态的定义对智能体非常关键，是人工智能要解决的一个基础性问题：如何描述智能体之外的世界（环境）？

状态是事物或智能体相关的环境的抽象表示，常用属性向量来表示各种关键属性，不重要细节或属性可以忽略不计。图3.7所示的八皇后问题的棋盘布局就是一个状态。棋盘中棋子偏移 1 毫米，对布局/状态没有影响。

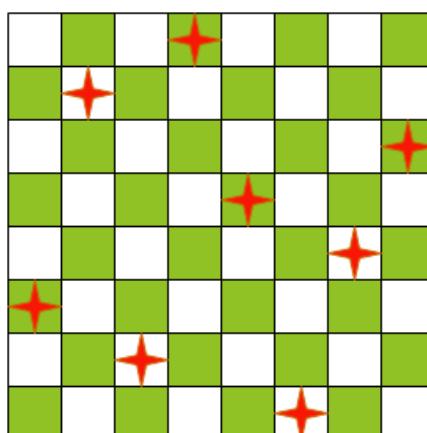


图 3.7: 八皇后问题的一个解

例 3.4 (三国华容道) 如图3.8、3.9所示的三国华容道游戏。



图 3.8: 三国华容道棋盘

赵云	曹操		张飞
马超	卒	卒	黄忠
超	卒	卒	
关羽			

图 3.9: 简化的三国华容道棋盘

三国华容道游戏中，五虎上将关羽、赵云、张飞、马超和黄忠带领四个卒子围捉曹操，拯救曹操，帮助曹操从棋盘下方的缺口逃离包围，移动时仅允许棋子往空白处移动。这是非常有名的中国传统益智游戏。

三国华容道游戏的任何一个棋盘可以定义为一个状态，类似图3.9，用一个二维的向量描述对应的棋盘；所有可能的棋盘就构成了三国华容道问题的状态空间。**注** 这个状态空间有多大？

例 3.5 (数独游戏) 如图3.10所示的数独游戏。数独盘面是个九宫，每一宫又分为九个小格。在这八十一格中给出一定的已知数字，在其他的空格上填入 1-9 的数字。使 1-9 每个数字在每一行、每一列和每一宫中都只出现一次。世界数独锦标赛比较谁最快完成数字的正确填写。

	C1	C2	C3	C4	C5	C6	C7	C8	C9
R1			8				2		
R2		3		8	2			6	
R3	7				9				5
R4		5						1	
R5			4				6		
R6		2						7	
R7	4				8				6
R8		7		1		3		9	
R9			1				8		

图 3.10: 数独游戏

数独起源于 18 世纪初瑞士数学家欧拉等人研究的拉丁方阵。20 世纪 70 年代，人们在美国纽约的一本益智杂志《Math Puzzles and Logic Problems》上发现了这个游戏。首届世界数独锦标赛于 2006 年在意大利卢卡举办，第八届于 2013 年在北京举办。

每个数独游戏的棋盘可被定义成一个状态；所有可能的棋盘就构成了数独游戏的状态空间。**注** 这个状态空间有多大？

状态空间中状态的数码通常随着状态描述向量维数而指数增加。这是状态定义时面临的最大问题，我们首先定义什么是“好的”状态。

知识点 3.6

thm:state-1 好的状态应该是满足问题约束条件的合法状态；好的状态空间中状态数目应尽可能少，尽可能把不可行或者从初态出发无法达到的状态移除。状态定义的好坏受到算法操作的限制，好状态应便于算法操作实现和高性能。



例题 3.2 (数码问题的状态空间) 图3.6所示的悬赏问题状态空间有多大?

解 若每个棋盘布局被定义为一个状态，那么棋盘的个数是一个排列问题，共有 $(n^2)!$ 种不同的棋盘。图3.6所示的棋盘 $n = 4$ ，状态空间包含大约 $16! \sim 2.09 \times 10^{13}$ 种不同的状态。

对于数码问题，当 $n = 3$ 时，状态空间的大小为 $9! = 362880$ ；当 $n = 5$ 时，状态空间大小为 $25! \sim 10^{25}$ 。

若把图3.6所示的悬赏问题里，从初态出发不可达的状态从状态空间中删除，那么状态空间大小能减少一半。

例题 3.3 (无解的悬赏) 证明图3.6所示数码问题无解。

解 将棋盘的 15 个数字从上到下，自左向右排列为一行并编号为 $1, 2, \dots, 15$ ，空白格子直接忽略，定义 $n_i, i = 1, 2, \dots, 15$ 为这 15 个数字的“逆序数”。所谓“逆序数” n_i 是指，在 15 个数字的排列中，数字 i 之后的格子中小于 i 的数字个数。棋盘上所有数字的逆序数之和为棋盘的逆序数。例如图3.5中， $n_1 \sim n_{15} = \{0, 0, 0, 0, 0, 0, 1, 1, 4, 0, 0, 0, 0, 0, 0\}$ ，所以图3.5的棋盘的逆序数为 $\sum_{i=1}^{15} n_i = 7$ 。

定义棋盘价值为“棋盘逆序数 + 空白格子所在行号”。对图3.6所示的棋盘，其中左边棋盘的价值为 4，右边棋盘的价值为 5。水平移动一个数字，棋盘价值的不变；上下移动一个数字，等价于在数字排列中前移三次或后移三次，只会改变移动时经过的三个数字的逆序数或移动数字的逆序数。例如，原来是 (小, 大) 的两个数，前移大的数字或后移小的数字，大数字的逆序数 +1，小数字的逆序数不变；原来是 (大, 小) 的两个数，前移小的数字或后移大的数字，大数字的逆序数 -1，小数字的逆序数不变；移动一次，改变一次棋盘逆序数的奇偶性；连续移动三次，棋盘逆序数的奇偶性发生改变；但是，空白格所在行号的奇偶性也发生改变，所以棋盘价值 (= 棋盘的逆序数 + 空白格的行号) 的奇偶性不变。

图3.6所示的两个棋盘，棋盘价值奇偶性不一致，故不存在从左边棋盘到右边棋盘的移动路径。

例 3.6 (n 皇后问题的状态空间) 给定一个 $n \times n$ 的方格棋盘，放置 n 个皇后，皇后间彼此不能攻击对方。若任意一种放置 n 个皇后的棋盘定义为一个状态，那么一共有 $\binom{n^2}{n} = \frac{(n^2)!(n^2-n)!}{n!}$ 种不同的状态。

如果我们要找到一个 n 皇后问题的解，可以采用穷举法。假设我们的计算机每秒能检查 10^9 个状态是否满足目标测试，当 $n = 8$ 时，约有 4.42×10^9 个状态，需要 4.42 秒找到解；当 $n = 10$ 时，约有 1.73×10^{13} 个状态，需要 480 个小时找到解； $n = 20$ 时，约有 2.78×10^{33} 个状态，需要 8.81×10^{16} 年找到解；而当我们对 n 皇后问题有个“好”的状态空间设计，算法也恰当，用现在的个人电脑，能在 1 分钟能找到 $n = 10^8$ 个皇后问题的解。由此可见，状态空间及其上的搜索算法的“良好”设计能极大加速问题的求解。

3.2.4 初态和目标测试

某些搜索问题，初态不同就意味着一个不同的搜索问题，例如三国华容道、数独游戏和数码问题等。

例 3.7 (三国华容道的初态) 每个不同的三国华容道的初始棋盘代表这一个新的游戏。如图3.11所示的三国华容道游戏的经典初始布局。

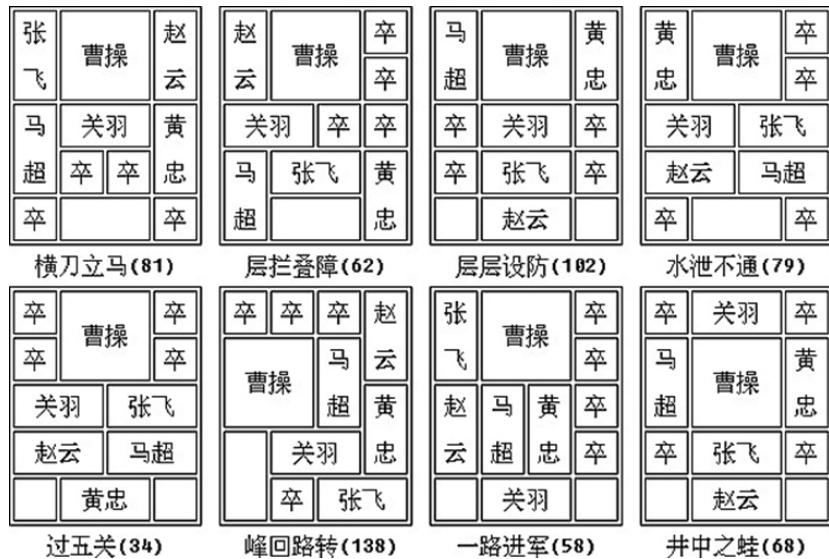


图 3.11: 三国华容道游戏的初始布局

三国华容道、数独游戏和数码问题等的初态被精确的指定。初态也可以不明确指定。比如 n 皇后问题，若初始时刻，随机在棋盘上放置 n 个皇后，即视其为初态，通过移动皇后，跳转到下一状态，最终到达合法的皇后布局状态。这种不明确指定初态的搜索，通常是指从任何一个初态开始搜索都可以，只要找到满足目标测试的状态即可。

目标测试是一组条件，这组条件给出了一个或多个目标状态。比如三国华容道游戏的目标测试是“曹操从棋盘下方逃离包围圈”，这个约束条件给出了一个由多个目标状态构成的集合。

3.2.5 后继函数

搜索问题的后继函数体现为状态图的边。状态图的边在实际问题建模过程中，常常被理解为“动作/行动”，表示一个状态在边表示的行动作用下，转换到了下一个状态。与我们讨论的表示智能体的函数相比，后继函数描述了一个状态到一个或多个状态的转换，这些状态都是同类型的。表示智能体的函数通常不要求输入和输出同类型。

例 3.8（数独问题的后继函数）如图3.12所示，数独问题的当前状态由灰色格子里的大写字母数字和白色格子构成。每个白色格子的左上角写出了一些小写数字，表示可以在当前白色格子里填写的可行数字。选择其中某个白色格子里的小写数字填写在所在白色格子里，就得到下一个状态。当前状态所有可能的下一状态可以由白色格子里左上角的小数字确定，一个小写的数字确定一个下一状态。

“选择一个小写数字并写作所在的白色格子里”这个行动，让数独问题从当前状态转移到了下一状态/后继状态。后继函数输入一个状态，返回 0 个、1 个或多个后继状态。在将实际问题建模为搜索问题并求解时，两个最重要的设计部件是状态和后继函数。

			1569	1456	45	13456	159	34569
8	2	7						
69	69		1	3	7	245	4568	2589
								24569
4	5		36	1269	16	8	1367	1279
16	67		2	4	156	57	9	3
								8
5	78		4	178	9	37	2	6
69			68	5678	2	57	57	4
3	4	35	257		8	6	13457	12579
	468		9	57	45	1	34567	578
2							8	34567
7	1		568	25	3	9	4568	258
								2456

图 3.12: 数独问题的后继函数

3.2.6 路径耗散

若后继函数表示执行一个行动/动作，那么我们可以把花费/代价标注在状态图的边上，看作后继函数的执行代价；也可以认为是一种“奖励”，执行完动作后获得的回报，但是此时最优解通常对应最大化行动序列获得的奖励之和。路径耗散一般是正的，我们通常假设路径耗散 c_{ij} 有一个有限的下界，即大于某个正常数 $c_{ij} \geq \epsilon > 0$ 。

例 3.9（路径耗散的例子）数码问题、 n 皇后问题和数独问题的路径耗散，如下：

- (1) 数码问题：每移动一次，代价为 1；
- (2) n 皇后问题：每放置/撤回一个皇后，代价为 1，从一个状态转移到下一个状态，花费的代价 $\leq 2n$ ；
- (3) 数独问题：填写/擦除一个数字，代价各为 1，从一个状态转移到下一个状态，花费的代价 $\leq 2n$ 。

3.3 应用问题的搜索形式化

如图3.13所示，利用计算机求解实际应用问题，建模和程序设计师两个关键步骤。数据结构更侧重在通过程序设计实现从逻辑模型到存储模型的转换；数据结构的设计应以程序设计实现的简洁和高效为指导。人工智能更多地讨论如何将现实世界建模为逻辑模型，将更复杂的现实问题转化为各种受限的、带约束的逻辑模型，以及这些逻辑模型对应的算法实现。从这个意义上讲，人工智能和数据结构一起完善了用计算机技术解决实际问题的流程。

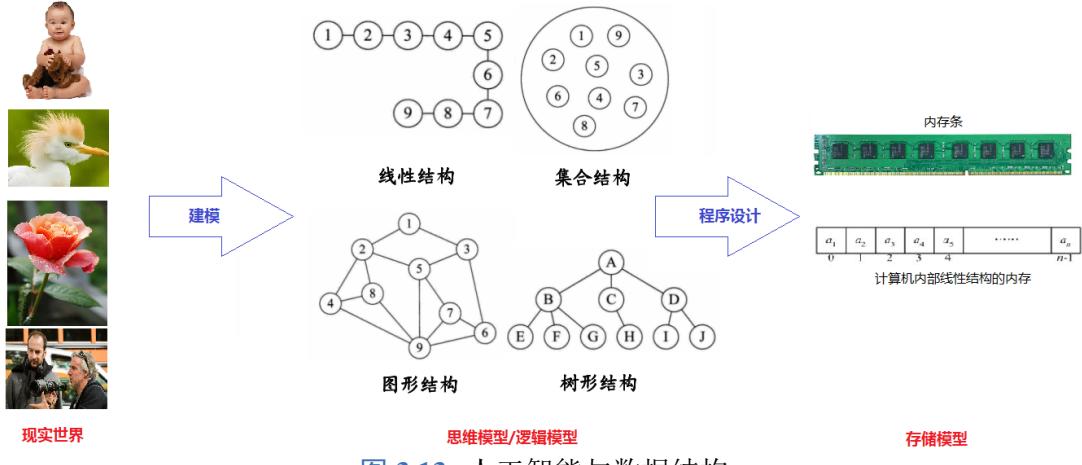


图 3.13: 人工智能与数据结构

应用计算机技术求解实际问题的第一步就是问题建模，也就是讨论如何将现实问题形式化为搜索问题，用搜索问题的五要素框架描述实际问题。

例 3.10（三国华容道问题的求解）第一步先要建模，定义三国华容道问题作为搜索问题的五要素，编码描述五要素，如图3.14所示，灰色格子表示空白，编码为一个 5×4 的数字矩阵；并设计后继函数（图中的连边），将状态图构造成初始状态为根的多叉树，并删除树上的重复状态；第二步，利用数据结构知识在线性结构的内存中编程实现，完成模型求解。如图3.14所示。

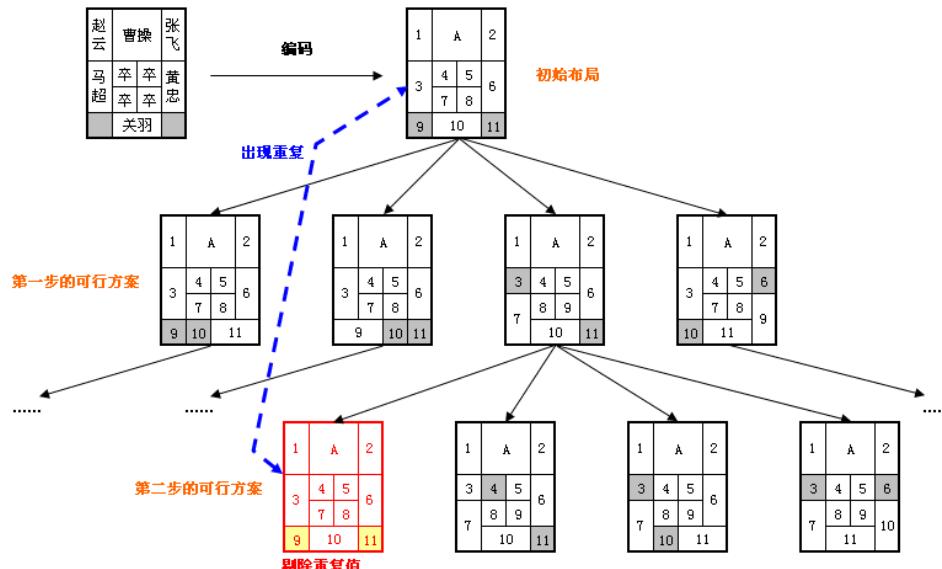


图 3.14: 三国华容道的建模及求解

将具体应用问题形式化为搜索问题，没有固定的方法；依赖于对问题的认知。我们通过三个例子来认知搜索问题的形式化技巧。

例 3.11（8 皇后问题的搜索形式化方法 1）五要素如下：

- (1) 状态与状态图空间：任何 0, 1, 2, ,8 个皇后在棋盘上时的布局代表一个状态；
- (2) 初始状态：0 个皇后在棋盘上的状态；
- (3) 目标测试：8 个皇后在棋盘上，彼此间相互攻击不到；
- (4) 路径耗散：放置/撤回在棋盘上的一个皇后，代价为 1；

(5) 后继函数：在一个空的格子上放置一个皇后，或撤回棋盘上的一个皇后，得到一个新状态。

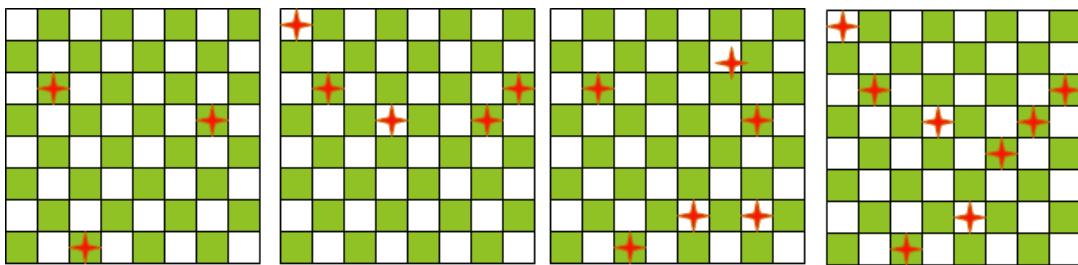


图 3.15: 8 皇后问题的形式化方法 1

这种 8 皇后问题的形式化方法 1 构成的状态图为高度为 9 的 64-叉树，包含约为 $64 \times 63 \times \dots \times 57 \sim 3 \times 10^{14}$ 种不同的状态顶点。每个状态有 64 种不同的后继状态，后继状态的数目是状态图顶点的（出）度，顶点的度越大，图通常越复杂。

例 3.12 (8 皇后问题的搜索形式化方法 2) 五要素如下：

- (1) 状态与状态图空间：任何 $0, 1, 2, \dots, 8$ 个皇后在棋盘左侧，且互不攻击时代表一个状态；所谓棋盘左侧互不攻击，就是前 k 列每对皇后，互不攻击；
- (2) 初始状态：0 个皇后在棋盘上的状态；
- (3) 目标测试：8 个皇后在棋盘上；
- (4) 路径耗散：放置一个皇后在棋盘上，代价为 1；
- (5) 后继函数：在 $k + 1$ 列放置第 $k + 1$ 个皇后到一个不受攻击的位置，得到一个新状态。

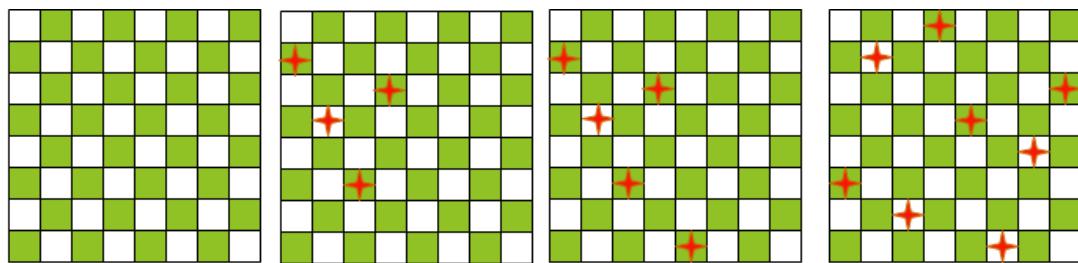


图 3.16: 8 皇后问题的形式化方法 2

8 皇后问题的形式化方法 2，构造的状态图只有 2057 个状态。然而这种方法并不能将 n 皇后问题完美解决，因为当 $n = 100$ ，形式化方法 2 得到的状态数目约为 10^{52} ，依然是一个非常巨大的状态空间。

例 3.13 (路径规划问题的形式化方法 1) 如图3.17所示的路径规划问题，可能是游戏地图或实际生活中机器人需要从初始位置（红色点标出）前进到目标位置（绿色点标出），图中咖啡色标出不可触碰的障碍物，寻找一条（最短）路径。

例 3.14 (路径规划问题形式化方法 1) 如图3.18 (a)所示，将地图网格化：令网格边长为 1，则对角线距离为 $\sqrt{2}$ ，机器人的状态用当前位置的坐标 (x, y) 来表示，所有的整数点坐标值构成状态空间；小方格的边为后继函数，后继状态是当前状态的东南西北四个正方向上，距离/路径耗散为 1 的下一个状态。

上述路径规划问题形式化方法 1 也可以在定义后继函数时包括四条 45 度对角线，即后继状态在当前状态周围最近的 8 个整数坐标位置上。此时，如图3.18 (b)所示，解是一个序列，相邻整数点/状态之间距离 $\leq \sqrt{2}$ ，如图3.18 (b)所示最短路径，仅仅是给定网格化的离散空间中的最优解。

例 3.15 (路径规划问题形式化方法 2) 如图3.19所示。

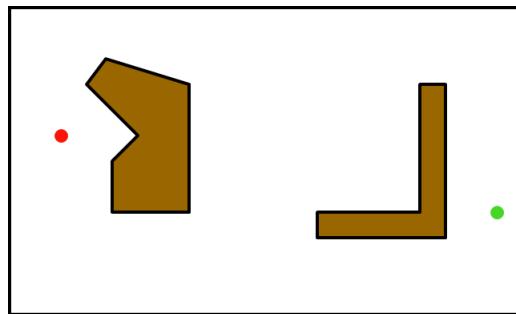
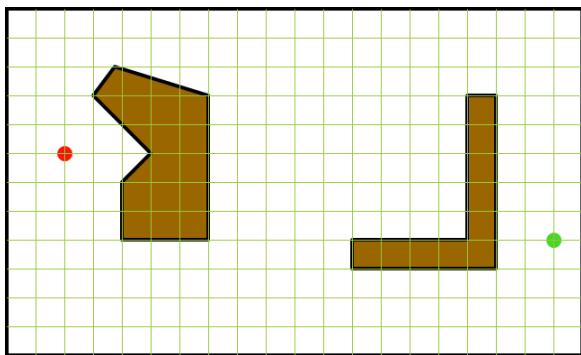
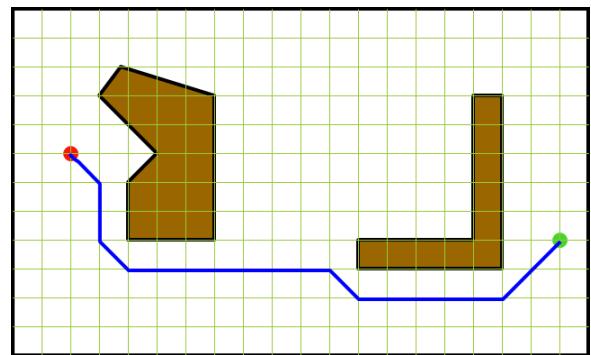


图 3.17: 路径规划问题



(a) 状态与后继函数



(b) 解与最短路径

图 3.18: 路径规划问题形式化方法 1

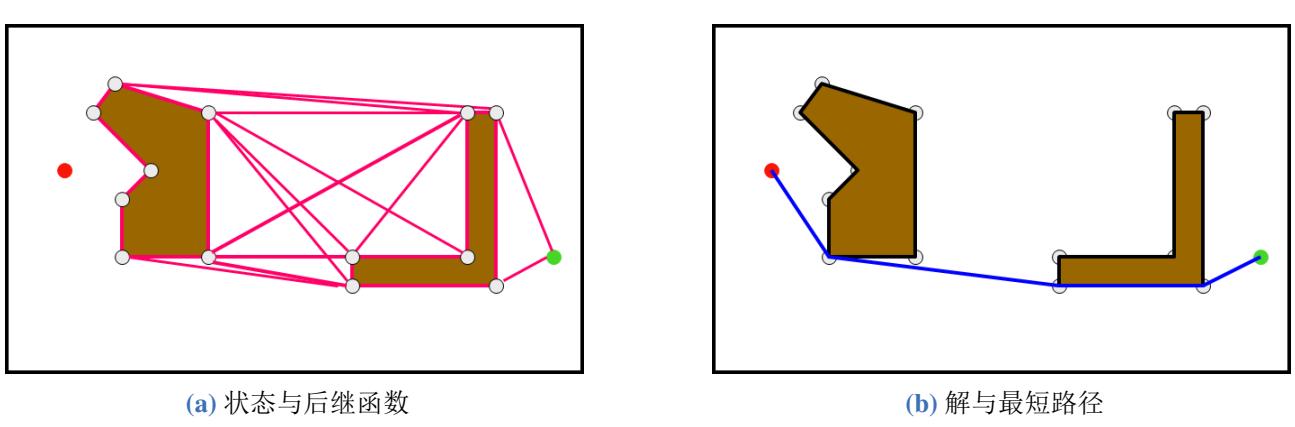
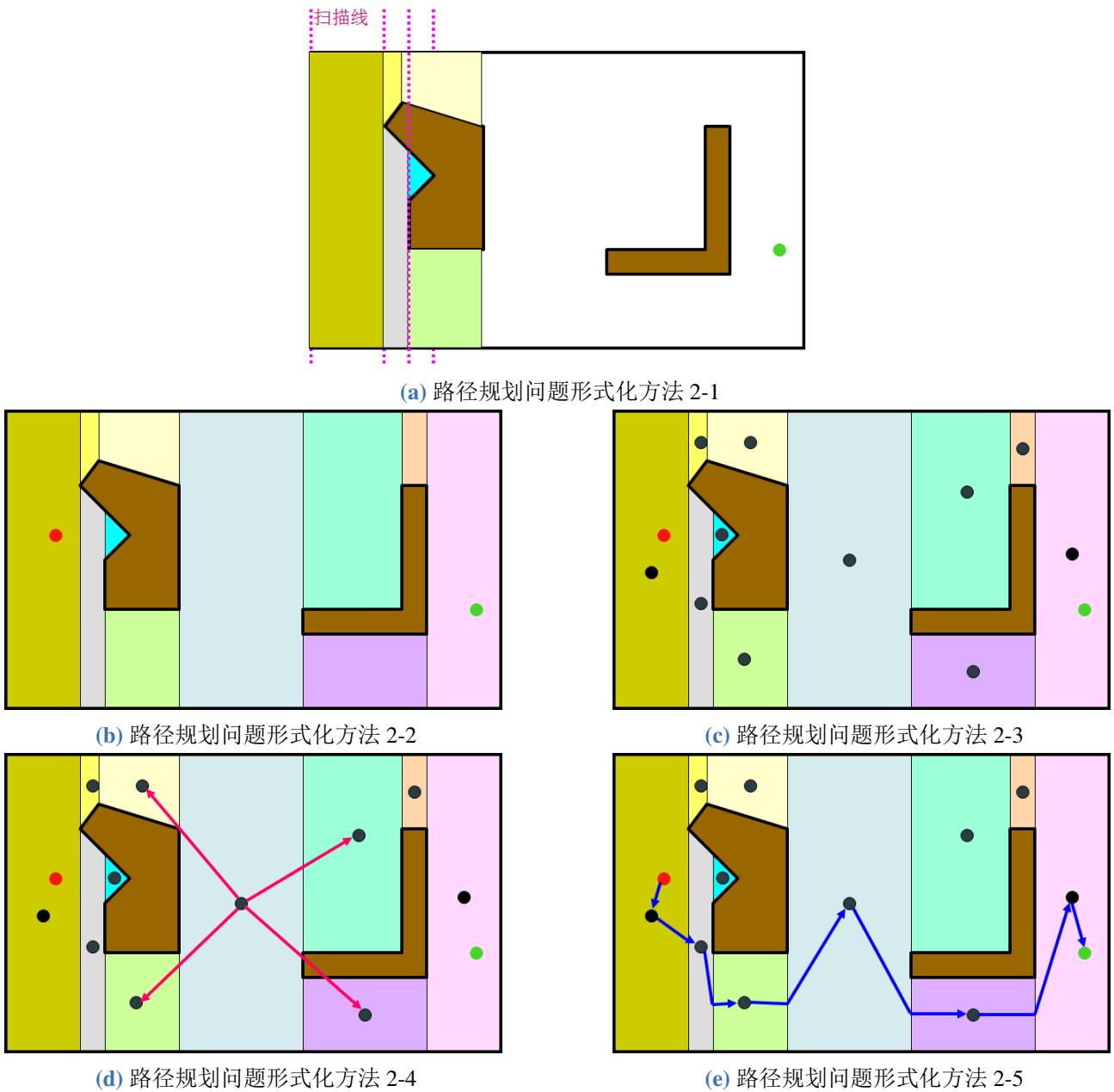
- (1) 假设垂直扫描线从左到右进行扫描；遇到障碍物的顶点（不妨设障碍物为多边形）则暂停，加标记；标记方法是对两次暂停之间的被扫描区域染上不同的颜色，如图3.19 (a)所示；
- (2) 被障碍物割裂的扫描线获得不同颜色的区域；整个地图扫描完毕后得到如图3.19 (b)所示的彩色着色图；
- (3) 将每个颜色块的“中心/重心”用黑色的点标记出来，用中心点代表该色块，如图3.19 (c)所示，共有 11 个色块，11 个中心点代表 11 个状态，加上初始位置和目标位置，状态空间一共由 13 个状态构成；
- (4) 后继函数：将有共边的色块中心点连上（无向）边，边两端的状态互为后继状态；如图3.19 (d)所示；
- (5) 定义路径耗散为边的长度，例如点之间的欧几里德距离；
- (6) 图3.19 (e)展示的解为一条路径。为了避开障碍物，如果两个有共边的相邻色块间的中心点连线经过障碍物，那么在共边上选择一个点加入状态空间。

路径规划问题形式化方法 2 也被称之为“扫描线法”，该方法得到的路径不一定是最短的，但是形式化生成的状态空间通常较小，特别是在障碍物较少或障碍物外形较简单的情况下。

例 3.16 （路径规划问题形式化方法 3）如图3.20 (a)所示。

- (1) 取障碍物（不妨设为多边形）的顶点以及初始地点和目标地点构成状态图的顶点；
- (2) 对任何顶点，连接其可视顶点（没有被障碍物阻挡），获得后继函数；
- (3) 路径耗散定义为边/弧的长度，顶点间的欧几里德距离；
- (4) 如图3.20 (b)所示，给出了一条最短的连续路径。

路径规划问题的形式化方法 3 也被称之为“障碍物边界法”，获得的通常是连续的最优解。状态空间的大小或状态图的复杂性取决于障碍物的边界的复杂性。



3.4 编程求解搜索问题

赫伯特·西蒙设计实验证明了人类解决问题的过程是一个搜索过程，我们了解了如何形式化一个具体的实际问题为搜索问题，接下来的任务是讨论如何设计算法，编程实现搜索问题的求解。

3.4.1 广度优先搜索算法

例 3.17（广度优先搜索）如图3.21 (a)所示有向图，红色顶点为初始状态，绿色顶点为目标状态，寻找从初始状态到目标状态的一条路径。执行广度优先搜索算法，记录搜索过程中经历过的顶点和边，在状态不重复访问的条件下，遇到目标状态时，停止搜索，并输出一棵搜索树，如图3.21 (b)所示。**注** 图3.21 (b)所示搜索树中标蓝色五角星的两个状态在重复访问时被丢弃，这棵搜索树多画出了这两个状态。

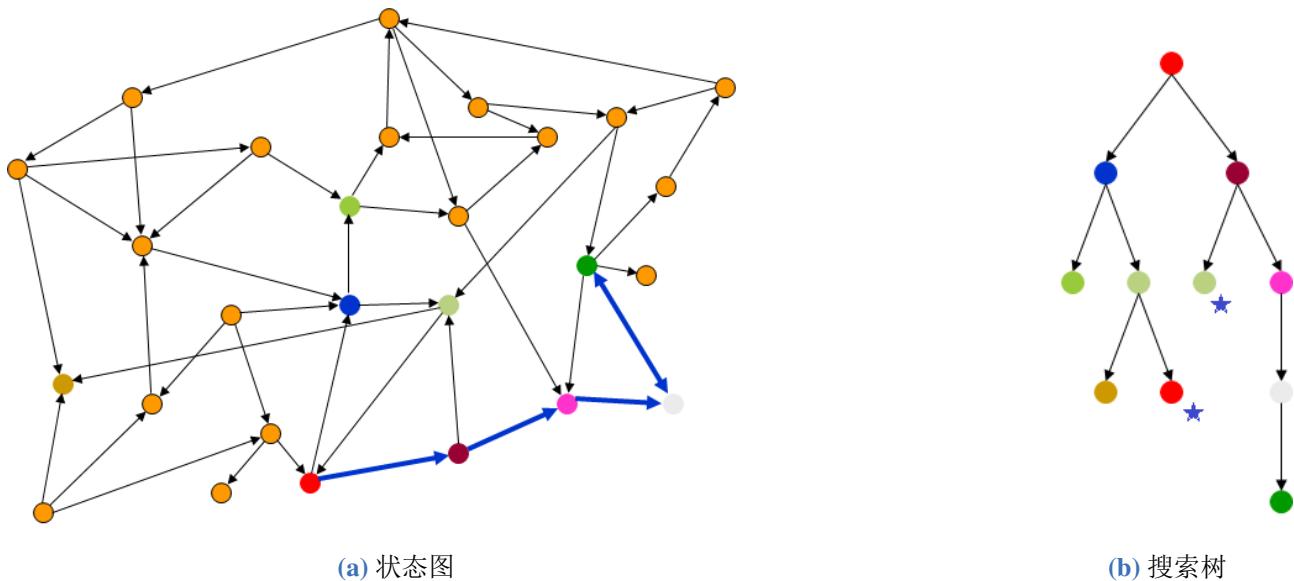


图 3.21: 广度优先搜索

广度优先搜索算法可用于各种类型的图的搜索问题求解，是一个通用的算法；特别地，对无权图来说，广度优先搜索能找到最短路径。采用邻接表存储状态图时，广度优先搜索算法的最坏时间复杂度是 $O(n + e)$ ，另需要 $O(n)$ 的内存空间来存储访问标记数组，避免状态的重复访问。

3.4.2 搜索算法的基准算法框架

编程实现广度优先搜索算法解决实际搜索问题时，首先要解决如何描述状态图顶点的问题。

知识点 3.7 (状态与节点)

状态图中的顶点描述了状态；而搜索过程中构造的搜索树的顶点我们称之为“节点”。一个状态可能对应多个节点，未访问时没有节点与之对应。



如图3.22所示的 8 数码问题求解时得到的搜索树的一个片段。搜索树的根（蓝色节点）和树最右下角的红色节点表示了状态图的同一个状态，即同一个棋盘布局；同一个状态在搜索树中可能有多个节点对应，这一情形在状态允许被重复访问时就会出现；可能还会导致另一个直接后果：搜索树的深度可能是无限的。

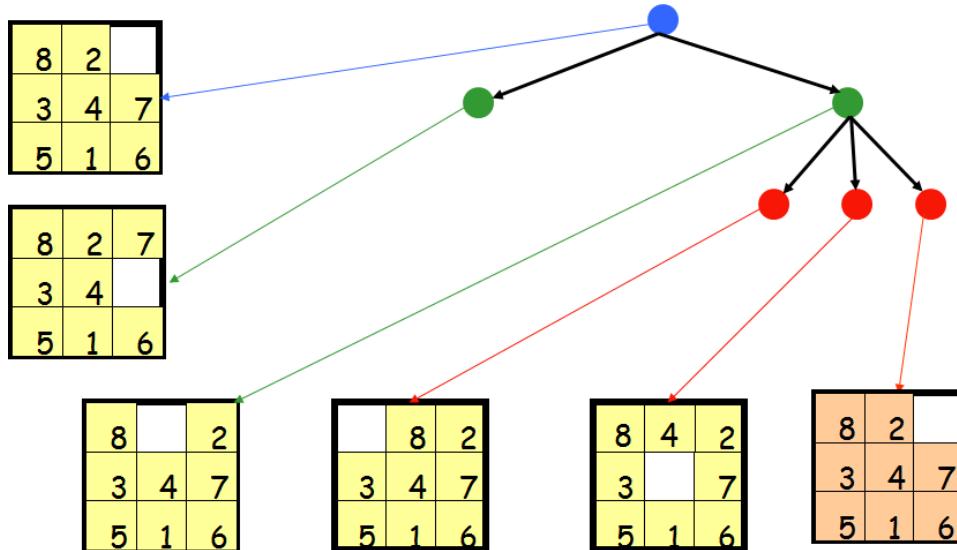


图 3.22: 状态与节点

知识点 3.8 (节点的数据结构设计)

节点是状态在搜索算法中的实例化表征，其结构设计如图3.23数码问题例子所示，包括的信息有：

- (1) 状态的向量，描述事物特征，如事物属性列表，如图3.23中数码问题的棋盘布局等；
- (2) 节点的深度，从根到节点的边的数量；通常设初始状态对应节点的深度为 0；
- (3) 构成搜索树的指针，如父节点指针，子节点指针等
- (4) 路径耗散，从根到节点的边上路径耗散之和；
- (5) 节点是否已扩展，“节点扩展”是针对节点的一个操作，若节点已经扩展，该标记为 `true`，否则为 `false`。

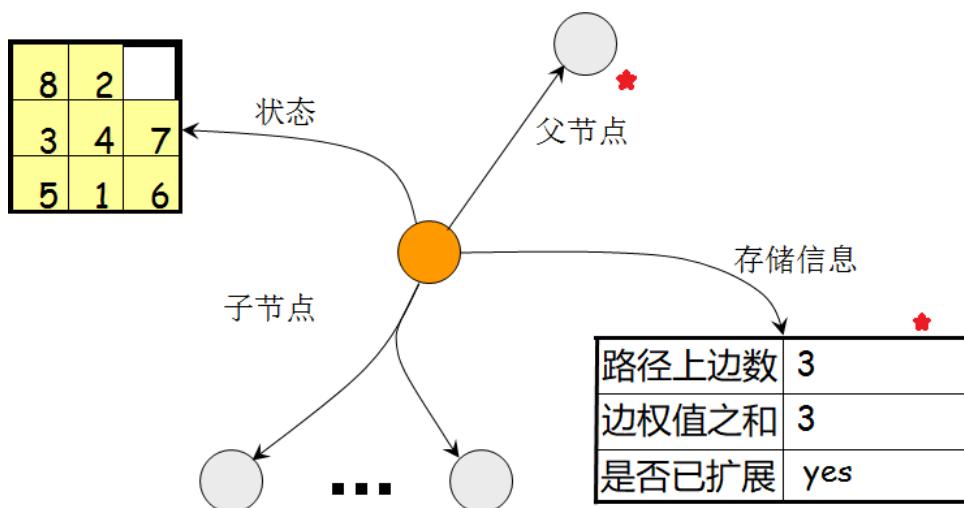


图 3.23: 节点的数据结构

状态图相当于数据结构中的逻辑结构；完成了节点的数据结构设计，我们还要先对涉及的几个关键操作和概念进行阐述，然后再给出搜索基准算法框架。

知识点 3.9 (节点扩展)

节点扩展直观理解就是把一个节点用它的后继节点（集合）来替换；对节点 N 的扩展，包括的处理有：

- (1) 填充节点 N 的信息，如搜索树的指针关系、路径长度、路径耗散、估计到目标状态的路径耗散等信息；
- (2) 评估状态图节点 N 的后继函数，对后继函数返回的所有后继状态，在搜索树上分别产生一个子节点与之对应。

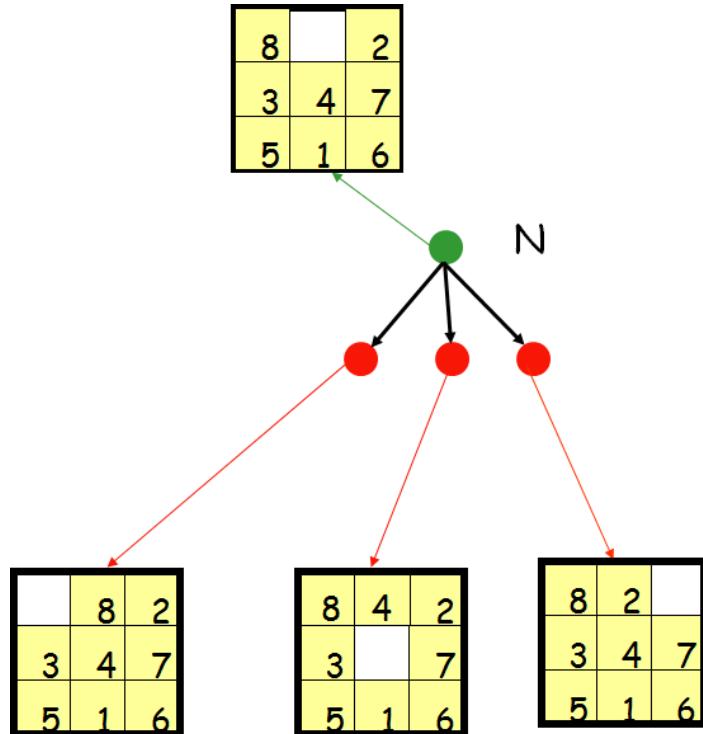


图 3.24: 节点扩展

节点扩展是搜索算法在执行中逐步生成搜索树的过程中最关键的操作。

知识点 3.10 (搜索的边界)

搜索过程中逐步构造搜索树上的节点，这些节点可划分为三个类：已访问过的节点、正准备访问的节点和未来可能要访问的节点。如图3.25所示，搜索边界就是正准备要访问的节点。

- (1) 图3.25中“云”中搜索树的那些节点；这些节点构成的集合记作 $Fringe$ ；
- (2) 搜索边界的节点构成了未扩展状态对应的节点集合，可理解为：在排队等待扩展的那些节点！
- (3) 搜索边界与叶节点不同，叶节点可能已经扩展过了，但是没有子节点；
- (4) 未扩展节点是搜索树上的节点，因为未扩展，所以还没有子节点，看上去“像”叶节点。



图3.25中，状态图中的顶点在搜索过程中构造对应的搜索树上的节点，其中云外的蓝色和绿色节点是已经访问过的节点；云中的节点是搜索边界 $Fringe$ 集合；对一个巨大的状态图 G 来说，还有大量的状态顶点对应的节点仍未在搜索树上构造出对应的节点。

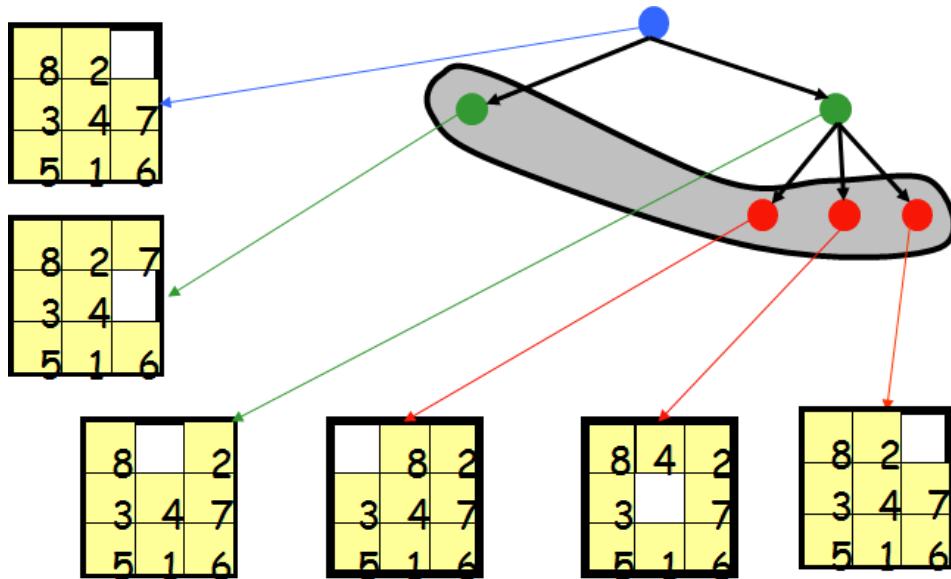


图 3.25: 搜索的边界

知识点 3.11 (搜索策略)

Fringe 中节点的优先次序就是搜索算法的搜索策略。



Fringe 集合是由已访问过/处理过状态的后继节点中未扩展的节点构成,通常用“队列”这种数据结构来存储 *Fringe* 中的元素。我们放弃用“队列”,重新定义一个 *Fringe* 节点的优先数组/有序数组(优先级随数字下标增大而降低),得到一个“搜索策略”。

在搜索策略采用 *Fringe* 集合的优先数组存储来实现时,对搜索策略有两个核心操作:插入一个节点到 *Fringe* 集合,从 *Fringe* 集合中移除一个节点。两个核心操作的定义如下。

知识点 3.12 (搜索策略的两个核心操作)

两个核心操作结束后要维持 *Fringe* 集合中的节点依然按优先次序存储。

- $INSERT(node, Fringe)$ 向优先数组 *Fringe* 中的“合适”位置插入节点 *node*;
- $node = REMOVE(Fringe)$ 从优先级数组中删除优先级最高的节点 *node*。



插入 $INSERT(\cdot)$ 操作要比较或计算待插入节点在优先数组中的位置,再完成插入;而移除 $REMOVE(\cdot)$ 操作直接删除优先数组的第一个元素。

知识点 3.13 (基准搜索算法的框架)

如算法3.1所示, 算法用了一个辅助数组 *Fringe*, 是一个存储未扩展节点的优先数组。

Algorithm 3.1: 搜索基准算法的框架

Input: G : 状态图; s_0 : 初态; 目标测试: $GOAL(\cdot)$;

Output: $path$: 代表解的路径

```

1  $path \leftarrow (s_0), Fringe \leftarrow \emptyset$                                 /* 初始化 */
2 if ( $GOAL(s_0) == T$ ) then
3   return  $path = (s_0)$ 
4 INSERT( $s_0, Fringe$ )
5 while  $T$  do
6   if  $empty(Fringe) == T$  then
7     return  $failure$                                 /* 返回  $failure$ , 表示无解 */
8    $N \leftarrow REMOVE(Fringe)$       /* 移除未扩展节点的优先数组首元素, 并记为  $N$  */
9    $s \leftarrow STATE(N)$                       /* 从节点  $N$  映射到状态  $s$  */
10  update  $path$ 
11  foreach  $s'$  in  $succ(s)$  do
12    为  $s'$  创建  $N$  的新子节点  $N'$ 
13    if  $GOAL(s') == T$  then
14      return  $(path, s')$                             /* 找到目标节点, 返回解 */
15    INSERT( $s', Fringe$ )

```



基准算法3.1中的第10~19行，描述了节点扩展；第18行的 $INSERT(\cdot)$ 需要评估节点的优先级，将节点插入到优先数组的“恰当”位置。

搜索问题求解的基准算法3.1和基于队列实现的广度优先搜索算法非常类似，有两个主要区别：

- (1) $Fringe$ 优先数组和队列类似；如果我们将 $Fringe$ 优先数组的插入 $INSERT(\cdot)$ 和移除 $REMOVE(\cdot)$ 操作重新定义为队列的入队和出队操作，那么基准算法就变成了广度优先搜索算法；
- (2) 因状态是否可重复访问，在不同的求解算法中有差别，所以基准算法略去了这部分代码，没有常见广度优先搜索算法中的“已访问节点标记数组”。

3.4.3 搜索算法的评估与分类

当我们把现实应用问题建模为搜索问题后，我们关注其中的 NP 或 NP-hard 类问题的求解，比如 TSP，数码问题等。一般认为这类问题不存在通用算法能在多项式时间内（时间复杂度是问题规模的多项式函数）求解出问题的所有实例（instances）。此时算法设计的意义在于：对问题的每个/类实例（instance）找到其最有效或尽可能高效的求解算法。

我们用从完备性、最优性和复杂性三个指标评估一个搜索算法求解一个搜索问题实例的性能。

知识点 3.14 (搜索算法性能评价的三个指标)

- (1) 完备性：问题有解时，算法能否保证返回一个解？问题无解时，算法能否保证返回 failure？
- (2) 最优性：能否找到最优解，返回路径耗散最小的路径？
- (3) 复杂性：时间复杂度用访问过的节点数目度量，空间复杂度用内存中同时保存的节点的最大数目来度量；搜索算法的时空复杂度和内存中构造的搜索树的结构密切相关：如树的分支因子，目标状态对应节点在搜索树的最浅深度等。而分支因子等搜索树的结构特征除了受算法（搜索策略）的影响，还受到状态空间、初态、后继函数的影响。



所有的搜索算法分为“有信息搜索”和“无信息搜索”两大类，搜索问题还有各种变型，变型问题有与问题相关的特点，依据问题特点发展了一些具有问题特色的搜索算法。如图3.26所示，我们将搜索问题的求解算法分成三大组，在不同章节讲述。

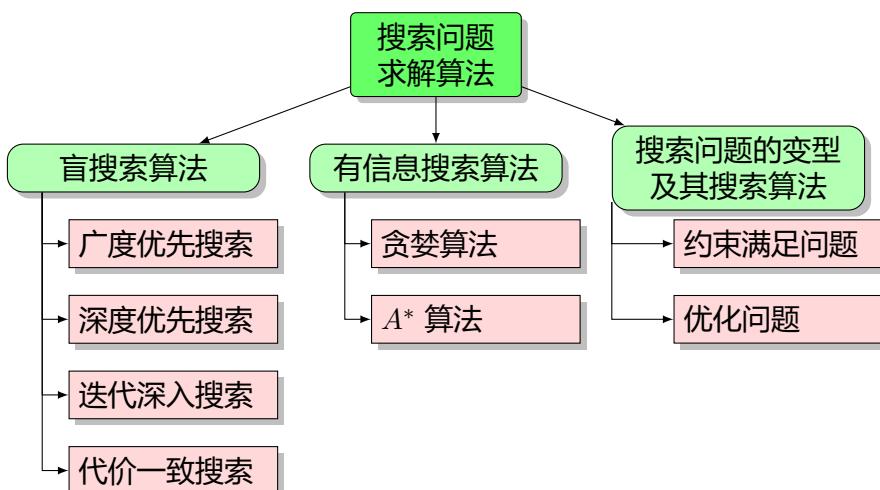


图 3.26: 搜索问题求解算法分类

定义 3.2 (盲搜索算法)

若搜索算法没有利用节点的状态等信息对搜索策略 *Fringe* 中的未扩展节点进行节点评价和基于评价的优先数组排序，那么该算法属于盲搜索算法或无信息搜索算法，简称“盲搜索”。

**定义 3.3 (启发式搜索算法)**

若搜索算法利用了已有路径、节点的状态和目标节点状态等信息对搜索策略 *Fringe* 中的未扩展节点进行节点评价和基于评价的优先数组排序，那么该算法属于启发式搜索算法或有信息搜索算法。



 第三章 习题

1. $n = 5$ 的数码问题，移动一个数字时，棋盘逆序数的奇偶性如何改变的？对一个 $N \times M$ 的大小的棋盘，定义数码问题，移动一个数字会对棋盘逆序数的奇偶性或棋盘价值的奇偶性有什么样的结论？
2. 如何实现算法3.1中第 11 行的“*update path*”功能？
3. 完善基准算法，将不允许状态重复访问的伪代码添加到代码框架中；并分析额外的时空开销。

第四章 盲搜索

内容提要

- 广度优先搜索
- 深度优先搜索
- 回溯法
- 深度受限深度优先搜索
- 迭代深入搜索
- 代价一致搜索
- 双向搜索

基准算法3.1中没有明确规定如何对 *Fringe* 优先数组的节点进行排序。若节点在 *Fringe* 优先数组中的次序是“无序”的，那么我们认为这类算法都是盲搜索。与之相对的，有信息搜索算法会将最有希望到达目标状态的节点定为最高优先级，排列在 *Fringe* 优先数组的最前端。

接下来，我们要讨论的一系列盲搜索算法与基准算法只在 *INSERT(·)* 操作的定义有差别，其它部分基本保持不变。

4.1 广度优先搜索

4.1.1 算法描述

知识点 4.1 (广度优先搜索)

若搜索的基准算法3.1中插入操作 *INSERT(·)* 总是将新节点插入到优先数组的末尾，即最低优先级位置，搜索的基准算法就变成了广度优先搜索。



4.1.2 算法评述

我们从完备性、最优性和复杂性三个方面评述广度优先搜索。假设在搜索问题的状态图中，顶点的最大（出）度为 b ，我们称 b 为“分支因子”；算法运行得到的搜索树中埋藏最浅的目标节点深度为 d 。 b 和 d 是评价搜索算法复杂性最重要的两个参数， b 由状态图的定义所确定， d 由搜索算法所确定。

完备性：若运行时间不限制，广度优先搜索在有解时能给出解；无解时能返回 *failure*，告知无解；因此广度优先搜索是完备的。

最优性：若状态图中每条边上的路径耗散相等，那么广度优先搜索在有解时一定返回路径耗散最小的最优解；因此广度优先搜索是最优的。

复杂性：算法的时间开销用算法访问过的节点数目 $l \leq 1 + b + b^2 + \dots + b^d = \frac{b^{d+1}-1}{b-1} \sim O(b^d)$ 来度量；空间复杂度用同时在内存中保存的最大节点数 s 来度量，搜索时构造的搜索树深度在不停增加，并保存在内存中，树“当前”的叶子节点的数目就是算法额外内存开销 *Fringe* 优先数组内存存储的节点数目，在接近搜索到目标节点，即搜索树的深度为 $d-1$ 或 d 时，*Fringe* 优先数组存储的节点数目约为 b^d 个，用大 O 记号，得到算法的空间复杂度为 $O(b^d)$ 。

例 4.1 (广度优先搜索的时空复杂度) 如表4.1所示，其中分支因子 $b = 10$ ，假设节点处理速度为每秒 10^6 个节点，每个节点占据 100 字节的内存空间。

d	访问节点数目	时间开销	空间开销
2	111	0.01 毫秒	11KB
4	11111	1 毫秒	1MB
6	$\sim 10^6$	1 秒	100MB
8	$\sim 10^8$	100 秒	10GB
10	$\sim 10^{10}$	2.8 小时	1TB
12	$\sim 10^{12}$	11.6 天	100TB
12	$\sim 10^{14}$	3.2 年	10000TB

表 4.1: 广度优先搜索的时空复杂度 $b = 10$

4.2 深度优先搜索

4.2.1 算法描述

深度优先搜索算法有些参考资料将其描述为递归算法或采用“栈”来实现。这里，我们通过修改表示搜索策略的 *Fringe* 优先数组来实现。

知识点 4.2 (深度优先搜索)

若搜索的基准算法 3.1 中插入操作 $INSERT(\cdot)$ 总是将新节点插入到优先数组的前端，即最高优先级位置，搜索的基准算法就变成了深度优先搜索。



4.2.2 算法评述

完备性：若运行时间不限制，搜索树是有限的，深度优先搜索在有解时能给出解；无解时能返回 *failure*，告知无解；因此，此条件下深度优先搜索是完备的。

最优性：不论状态图每条边上的路径耗散是否相同，深度优先搜索返回的解不一定是最优的；因此深度优先搜索无法确保最优性。

复杂性：深度优先搜索算法的时间开销用算法访问过的节点数目 $l \leq 1 + b + b^2 + \dots + b^m = \frac{b^{m+1}-1}{b-1} \sim O(b^m)$ 来度量，其中 m 是搜索树上叶子节点的最大深度，其值通常远大于埋藏最浅的目标节点的深度 d ；空间复杂度用同时在内存中保存的最大节点数 s 来度量，搜索时构造的搜索树深度在不断增加，但是每次扩展节点时，*Fringe* 优先数组为搜索树的每一层至多保存 b 个节点，保存的节点如果深度相同则有共同的父节点；又因为搜索树的高度最多为 m ，所以算法额外内存开销 *Fringe* 优先数组中存储的节点数目至多为 bm ，用大 O 记号，得到算法的空间复杂度为 $O(bm)$ 。

深度优先搜索算法相较于广度优先搜索算法，降低了内存需求，但是增加了时间开销，丧失了最优性的保证。

4.3 回溯法

4.3.1 算法描述

回溯法是对深度优先搜索的改进，其基本框架遵循深度优先搜索，仅在用深度优先搜索算法扩展节点时，从节点对应状态返回的多个后继状态中随机选择一个未访问过的状态，扩展出一个搜索树上的子节点，达到节省内存使用的目的。

知识点 4.3 (回溯法)

在深度优先搜索的过程中，每次扩展节点时，最多只选择一个后继状态，扩展出一个子节点；最多同时保存 $O(m)$ 个节点，以节省内存，其中 m 是搜索树的最大深度。



4.3.2 算法评述

完备性：同深度优先搜索。

最优性：同深度优先搜索。

复杂性：时间复杂度同深度优先搜索，空间复杂度为 $O(m)$ ，因为 *Fringe* 优先数组只为搜索树的每一层保存一个节点。

回溯法相较于广度优先搜索和深度优先搜索，进一步降低了内存需求，但是实现上有一个小问题，参考本章习题 1。

4.4 深度受限深度优先搜索

4.4.1 算法描述

深度优先搜索相较于广度优先搜索，一开始可能就会沿着搜索树的某个分支“深入”到树的“底部”，此时如果找到一个目标状态，很难保证目标状态是埋藏最浅的。因此，限制深度优先搜索过程中节点扩展的深度成为一种思路。

知识点 4.4 (深度受限深度优先搜索)

在深度优先搜索的过程中，当节点的深度大于预设置的阈值 k 时，节点不再扩展。



4.4.2 算法评述

完备性：不能保证完备性。若 $k < d$ ，此时有解，但是算法返回 *failure*，表示无解。

最优性：同深度优先搜索。例如埋藏最浅的目标节点深度为 $d = 3$ ，而 $k = 10$ ，假设深度优先搜索找到了搜索树上另一个分支上深度为 7 的一个目标节点，此时的找到路径不是最优的。若 $k = d$ 或 $k > d$ 时，没有深度为 $d + 1, d + 2, \dots, k - 1$ 的目标节点，那么算法能返回最优解。

复杂性：复杂性类似深度优先搜索，将 m 替换为 k ，得到时间复杂度为 $O(d^k)$ ，空间复杂度为 $O(bk)$ 。

深度受限深度优先搜索的关键在于设置一个合适的阈值 k 。

4.5 迭代深入搜索

4.5.1 算法描述

为解决深度受限深度优先搜索算法阈值 k 设置的问题，我们给出一个穷举阈值 k 的方案。

知识点 4.5 (深度受限搜索)

令 $k = 0, 1, \dots$, 不断调用深度受限深度优先搜索。



4.5.2 算法评述

完备性: 若运行时间足够, 当有解且 $k = d$, 会返回找到的解; 当无解时, 算法返回 *failure*, 表示无解。

最优性: 在当单步路径耗散相同的条件下, 当 k 逐步增加到 d 时, 返回最优解。

复杂性: 对每个 $k \leq d$ 的值, 算法至多访问一个高度为 k 的 b 叉完全树, 即 $1 + b + b^2 + \dots + b^k = \frac{b^{k+1}-1}{b-1}$ 个节点, $k = 0, 1, 2, \dots, d-1, d$, 对访问过的节点数按不同的 k 求和, 得到时间复杂度 $(d+1)(1) + db + (d-1)b^2 + (d-2)b^3 + \dots + (1)b^d \sim O(b^d)$, 空间复杂度为 $O(bd)$ 。

从时空复杂度, 完备性和最优性三个评价指标上看, 迭代深入搜索同时获得了广度优先搜索和深度优先搜索二者的优势。尽管迭代深入搜索对 $k < d$ 时的搜索花费了“大量无用”的时间代价, 但是迭代深入搜索和广度优先搜索的时间开销在量级上是一样的, 都是 $O(b^d)$ 。

例 4.2 (广度优先搜索和迭代深入搜索的时间开销比较) 如表4.2所示, 列出来两种算法在搜索树每层上访问的节点数, 以及最后一行求出了访问节点的总数。

深度	$d = 5, b = 2$		$d = 5, b = 10$	
	广度优先搜索	迭代深入搜索	广度优先搜索	迭代深入搜索
0	1	$1 \times 6 = 6$	1	6
1	2	$2 \times 5 = 10$	10	50
2	4	$4 \times 4 = 16$	10	400
3	8	$8 \times 3 = 24$	1000	3000
4	16	$16 \times 2 = 32$	10000	20000
5	32	$32 \times 1 = 32$	100000	100000
访问节点总数	63	120	111111	123456

表 4.2: 广度优先搜索和迭代深入搜索访问节点数对比

对大多数的搜索问题来说, 埋藏最浅的解的深度通常是未知的, 由表4.2可知, 随着分支因子 b 的增大, 迭代深入搜索相对于广度优先搜索付出的额外的时间代价比例会降低, 因此, 为降低内存开销, 采用迭代深入搜索成为盲搜索算法的首选。

4.6 代价一致搜索

4.6.1 算法描述

若将迪杰斯特拉算法适配到搜索基准算法框架中，那么得到代价一致搜索算法。

知识点 4.6 (代价一致搜索)

Fringe 优先数组优先级排序规则：按照节点数据结构中边权值之和（路径耗散）由小到大排，路径耗散最小者优先级最高；路径耗散最大者优先级最低。



4.6.2 算法评述

完备性：若分支因子 b 是有限的，单步路径耗散有正的下界，即 $c_i \geq \epsilon > 0$ 时，是完备的。[注](#) 证明略。

最优性：代价一致搜索是由路径耗散引导搜索过程，第一次找到目标状态的路径一定是路径耗散最小的，能确保最优性。

复杂性：时空复杂度和单步路径耗散的下界 ϵ 相关，为 $O(b^{\lceil C^*/\epsilon \rceil})$ 。[注](#) 证明略，有兴趣了解自行查看相关文献。

当边权值（单步路径耗散）不完全相同时，代价一致搜索就是迪杰斯特拉算法；当边权值（单步路径耗散）相同时，代价一致搜索等价于广度优先搜索算法。

4.7 双向搜索

4.7.1 算法描述

双向搜索不同于我们前面介绍的基准算法的各种变型，而是一种组合策略，通过重复运行上述搜索算法的变型获得问题的解。

知识点 4.7 (双向搜索)

分别从初态和目标状态出发，启动两个搜索算法（正向搜索和反向搜索），各维护一个 *Fringe* 集合，分别记录从初态和目标状态开始搜索的未扩展节点集合；当两个集合相交时，算法结束。



4.7.2 算法评述

完备性：受正向搜索和反向搜索算法的影响，正反向搜索算法有一个能确保完备性，算法即是完备的。参考本章习题 2。

最优性：同样由正反向搜索算法的最优性确定。参考本章习题 3。

复杂性：如果正反向搜索算法（采用与广度优先搜索）的同时并行执行，且分支因子 b 一样，那么时间复杂度约为 $O(b^{d/2}) \ll O(b^d)$ 。尽管时间复杂度依然分支因子 b 的指数函数，但是在实际应用中，可使用的范围获得了扩展。双向搜索更一般的时间复杂度和空间复杂度的讨论需要针对特定的正反向搜索算法来展开。

双向搜索改变了搜索基准算法的框架，并非一个“原子”的算法，属于组合类算法。双向搜索的正向搜索和反向搜索算法可以相同也可以不同。在实际应用中，如果能采用双向搜索，通常能带来时间性能上的明显提升。

但是，双向搜索存在一个困难：如何找到合适的反向搜索算法？具有唯一目标状态的，或者能找到一个精确目标状态，且能将后继函数变换成“前驱函数”搜索问题，比较容易找到反向搜索算法。

知识点 4.8 (盲搜索算法的比较)

各评价指标的值如表4.3所示，其中 b 是分支因子； d 是埋藏最浅的目标节点的深度； m 是搜索树叶子的最大深度； k 是深度阈值。

评价标准	广度优先	迭代深入	深度优先	深度有限	代价一致	双向搜索
完备性	是 ¹	是 ¹	是 ¹	否	是 ^{1,2}	是 ^{1,4}
时间复杂度	$O(b^{d+1})$	$O(b^d)$	$O(b^m)$	$O(b^k)$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^{d/2})$
空间复杂度	$O(b^{d+1})$	$O(bd)$	$O(bm)$	$O(bk)$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^{d/2})$
最优性	是 ¹	是 ³	否	否	是	是 ^{3,4}

¹ 若 b 有限，则完备；

² 若单步路径耗散有正的下界，即 $c_i \geq \epsilon > 0$ ，则完备；

³ 若单步路径耗散相同，则最优；

⁴ 若正反向搜索都采用广度优先搜索

表 4.3: 盲搜索算法的比较

4.8 状态的重复访问

知识点 4.9 (状态被重复访问的原因)

若状态图中的边是无向的，或者存在环路，那么状态可能被重复访问。

例如三国华容道游戏、数码问题等，游戏的行动可以“撤回”，返回前一状态。而八皇后问题的形式化方法2，状态图是一颗高度为9的8叉树，且边是有向边（放置皇后的动作不可撤回），状态不可能被重复访问。

如果描述搜索问题的状态图中的顶点在访问过程中被重复访问，可能会导致在搜索树中有多个节点与同一个状态对应，带来搜索树高度无限增加等问题。因此，我们需要一些额外的技术来避免状态的重复访问。

知识点 4.10 (避免状态的重复访问：方法一)

为每个状态设置一个访问标记 $v_i = \{true, false\}$ ，其中 $\{v_i, i = 1, 2, \dots\}$ 是状态图的顶点集；搜索开始时初始化所有状态的访问标记为 $false$ ，搜索过程中，若有一个访问标记为 $false$ 的状态，在其搜索树上对应节点第一次被构造出来时，设置该状态的访问标记为 $true$ 。

避免状态重复访问方法一简单直观，但是当状态图的顶点数目非常庞大时，例如达到量级 $O(2^n)$ ，可能无法在内存中分配对应大小的访问标记集合。

我们知道在大规模状态图上执行搜索算法算法时，状态图的状态可以被分为已被访问和扩展过的

状态，已被访问但未扩展过的状态和其它状态三类。前两类状态对应着搜索树上的节点，保存在内存中；第三类“其它状态”可能包括了状态图的绝大多数状态，内存无法容纳下它们。因此，我们通过只保存前两类状态，避免状态的重复访问。

知识点 4.11 (避免状态的重复访问：方法二)

设置一个 *CLOSED* 表保存已访问且扩展过的状态；设置一个 *OPEN* 表保存已访问但未扩展过的状态；当一个状态准备扩展时，若已保存在 *CLOSED* 表中，则丢弃该状态。



避免状态重复访问的方法二用 *CLOSED* 表和 *OPEN* 表保存了状态图的一个“很小”的部分，注意到保存的是状态的“标识”，而非描述是否访问过的 $\{true, false\}$ 标记。

例 4.3 代价一致搜索：避免状态的重复访问 具体步骤如下：

- 1 设 *CLOSED* 表中保存了已扩展的所有状态及其最优路径耗散（从初态出发）；
- 2 若状态 $s \in \text{CLOSED}$ ，则检查 s 的路径耗散，并和“当前路径 + s ”的总路径耗散进行比较，取较小的路径耗散者，更新状态 s 的路径耗散；
- 3 若 $s \notin \text{CLOSED}$ 表中，则放入 *CLOSED* 中，并开始扩展：将 s 的所有不在 *CLOSED* 表中的后继状态放入 *OPEN* 表中；
- 4 重复上两步；

知识点 4.12 (树搜索和图搜索)

在搜索过程中，若允许重复访问同一个状态，不同的实现有不同的结果。

- (1) 若一个状态的每次访问都构造一个新的搜索树节点，那么就会形成从初态（搜索树的根）出发的多条路径，此为“树搜索”，搜索的过程中伴随搜索树的扩张；
- (2) 若只有一个状态的第一次访问会构造一个新节点，其它访问仅添加到该状态对应节点的连边，此为“图搜索”，搜索过程中伴随着状态图被逐步“复现”；
- (3) 若只有一个状态的第一次访问会构造一个新节点，其它访问仅在到达该状态对应节点路径耗散更小时，添加到节点的连边，并删除原有到节点的连边，此时依然是“树搜索”，搜索过程中伴随着搜索树的扩张。



状态是否允许被重复访问关系到搜索的完备性，有如下结论。

知识点 4.13 (状态被重复访问与搜索的完备性)

- (1) 若状态空间是无限的，一般来说，搜索是不完备的；
- (2) 若状态空间有限，但允许状态被任意次重复访问，搜索一般是不完备的；
- (3) 若状态空间有限，访问时重复访问的节点被丢弃，则搜索是完备的，但是一般不是最优的。



 第四章 习题

1. 深度优先搜索在扩展节点时，会将一个状态的所有后继状态表示成节点存入 *Fringe* 优先数组中，但是回溯法只选择一个后继状态表示并存入，如何避免这个状态的同一个后继状态被重复多次选中并存入 *Fringe* 中？如何编程实现这个功能？开销是什么？
2. 证明双向搜索中，正反向算法中有一个是完备的，那么该双向搜索算法是完备的。
3. 说明双向搜索算法在不同正反向搜索算法的最优性条件下的最优性。

第五章 启发式搜索

内容提要

- 最佳优先搜索
- 启发式函数
- 设计启发式函数
- A^* 算法

优先构建更有希望找到目标节点搜索树分支进行搜索，将会极大地减少搜索过程中探索的无效分支，提高算法性能。最佳优先搜索的思想是启发式搜索的基础。

5.1 最佳优先搜索

定义 5.1 (最佳优先搜索)

即采用搜索策略：从搜索边界的优先数组 $Fringe$ 中选择最佳节点进行扩展。



最优优先搜索要解决两个问题：

- (1) 什么是最佳节点？
- (2) 如何获得最佳节点？

注 “最佳”是指在每个（准备扩展节点的）时刻所能找到的最佳扩展节点；最佳扩展节点通常是随着搜索的进行在不停地变动。

知识点 5.1 (节点评估)

对 $Fringe$ 优先数组中的节点 N ，设计非负状态评估函数 $f(N)$ ，表示从一个状态节点到另一个状态节点的路径耗散的大小。有三种节点评估方法：

- (1) 从初始状态节点到当前状态节点 N 的路径耗散；
- (2) 从当前状态节点 N 到某个目标状态节点的路径耗散；
- (3) 从初始状态节点出发到达当前节点 N ，然后前往某个目标状态节点的路径耗散。



知识点5.1定义了三种节点评估的方法，其中方法(1)没有使用节点中保存的状态信息，只利用了已经保存在节点中的路径耗散信息，基于方法(1)的搜索即代价一致搜索，属于盲搜索；而节点评估方法(2)和(3)利用了节点 N 中保存的状态信息，通常会估计节点 N 和某个目标节点之间的路径耗散，例如数码游戏中，利用当前节点对应的棋盘状态和目标棋盘状态之间的差异信息估计路径耗散，故基于节点评估方法(2)和(3)的搜索属于有信息搜索。

5.2 启发式函数

定义 5.2 (启发式函数)

从当前节点 N 到某个目标状态节点的路径耗散的估计值，记作 $h(N)$ 。



定义 5.3 (节点评估函数)

三种节点评估方法对应三类节点评估函数 $f(N)$ 。

- (1) $f(N) = g(N)$: $g(N)$ 表示从初始状态节点到当前状态节点 N 的路径耗散，已经保存在当前节点的数据结构中；
- (2) $f(N) = h(N)$: 启发式函数，估计值；
- (3) $f(N) = g(N) + h(N)$: 从初始状态节点到当前状态节点 N 的路径耗散加上当前状态节点 N 到达目标状态节点的估计路径耗散之和。

**定义 5.4 (启发式搜索)**

搜索基准算法中采用节点评估函数 $f(N) = h(N)$ 和 $f(N) = g(N) + h(N)$ 为搜索策略 *Fringe* 优先数组排序的搜索。



例 5.1 (贪婪搜索算法) 在启发式搜索中，若节点评估函数为 $f(N) = h(N)$ ，则搜索被称为“贪婪搜索算法”。

例 5.2 (A^* 算法) 在启发式搜索中，若节点评估函数为 $f(N) = g(N) + h(N)$ ，则搜索被称为 A^* 算法。

例题 5.1 (机器人导航问题) 如图5.1所示，方格地图中，每个格子代表一个状态，机器人从红色格子出发，前往目的地绿色格子，其中灰色格子表示障碍物，不可通行。

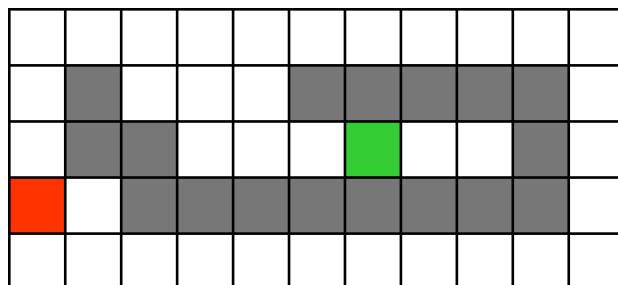


图 5.1: 机器人导航问题

解 (贪婪搜索算法): 采用节点评估函数 $f(N) = h(N)$ ，其中 $h(N)$ 表示红绿格子之间的曼哈顿距离(或城市区块距离，L1 距离)。如图5.2所示，在所有非障碍物的格子里标出了格子距离目标格子(绿色格子)的曼哈顿距离。图5.3中的蓝色格子给出了采用贪婪搜索时，机器人访问过的、最多的可能状态。

8	7	6	5	4	3	2	3	4	5	6
7		5	4	3						5
6			3	2	1	0	1	2		4
7	6									5
8	7	6	5	4	3	2	3	4	5	6

图 5.2: $f(N) = h(N)$ ，不同格子的 $f(N)$ 的值

8	7	6	5	4	3	2	3	4	5	6
7		5	4	3						5
6			3	2	1	0	1	2		4
7	6									5
8	7	6	5	4	3	2	3	4	5	6

图 5.3: $f(N) = h(N)$ ，机器人访问过的格子

(A^* 算法): 采用节点评估函数 $f(N) = g(N) + h(N)$ ，其中 $h(N)$ 表示红绿格子之间的曼哈顿距离，如图5.4所示，白色格子里是 $h(N)$ 的值，灰色格子里是 $h(N) + g(N)$ 的值，灰色格子和蓝色格子是机器人访问过的状态。

8+3	7+4	6+3	5+6	4+7	3+8	2+9	3+10		4	5	6
7+2		5+6	4+7	3+8							5
6+1			3	2+9	1+10	0+11	1		2		4
7+0	6+1										5
8+1	7+2	6+3	5+4	4+5	3+6	2+7	3+8	4	5	6	

图 5.4: $f(N) = g(N) + h(N)$

比较求解例题5.1的贪婪搜索算法和 A^* 算法，可以发现 A^* 算法访问过的状态数目更少。对于一个未知问题，采用 A^* 算法通常是更有效的。

5.3 启发式函数的设计

节点评估函数的设计关键在于启发式函数的设计，启发式函数的设计与具体问题（问题的状态描述）相关，通常存在多种启发式函数的设计方案。

例 5.3（路径规划问题的启发式函数）如图5.5所示路径规划问题，网格点构成状态集合，单位长度的边指示状态之间的后继函数。若绿色点 g 是目标状态，设计启发式函数估计当前状态 N 到目标状态 g 的路径耗散 h 。

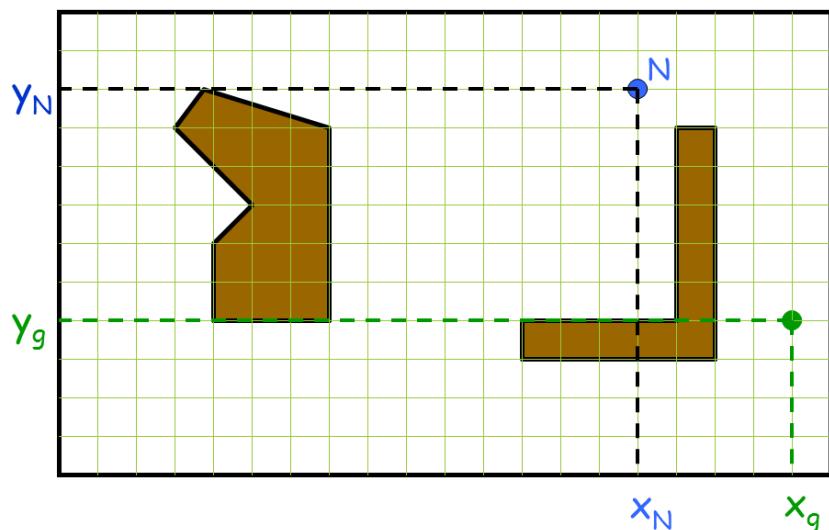


图 5.5: 路径规划问题

有多种启发式函数的定义方法，这里给出两种。

(欧几里得距离) : $h_1(N) = \sqrt{(x_N - x_g)^2 + (y_N - y_g)^2}$, 如图5.6所示

(曼哈顿距离) : $h_2(N) = |x_N - x_g| + |y_N - y_g|$, 如图5.7所示

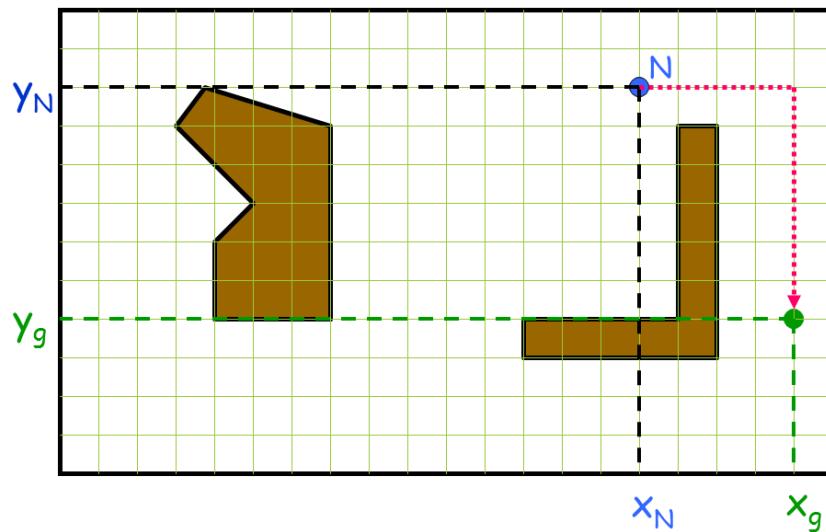


图 5.6: 路径规划问题

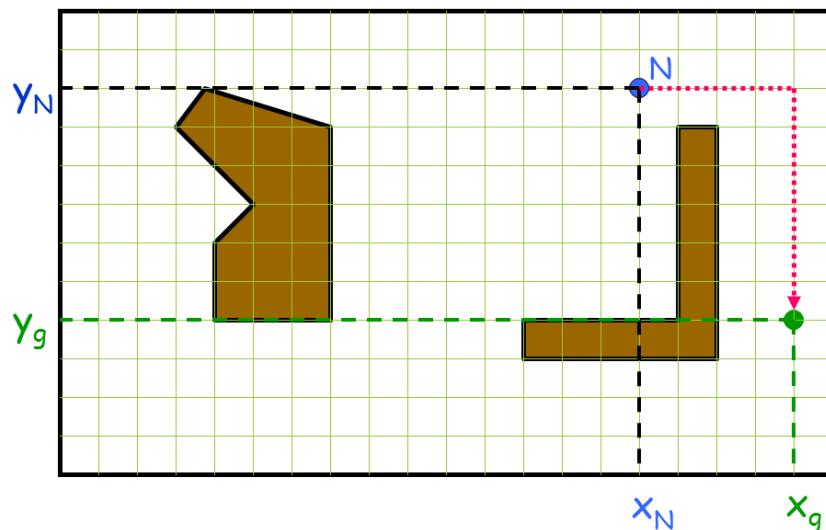


图 5.7: 路径规划问题

例题 5.2 (8 数码问题的启发式函数) 如图5.8所示的 8 数码问题, 给定了如图5.8 (a)所示的初始状态 N 和如图5.8 (b)所示的目标状态 $Goal$, 请设计启发式函数 $h(N)$ 。

5		8
4	2	1
7	3	6

(a) 当前状态

1	2	3
4	5	6
7	8	

(b) 目标状态

图 5.8: 8 数码问题: 基于当前状态和目标状态设计启发式函数

解 比较将当前状态和目标状态, 依据比较的方法不同, 我们给出如下三种启发式函数:

- (1) $h_1(N) = \text{数字错误放置的格子数} = 6$
- (2) $h_2(N) = \text{所有数字到其正确位置的曼哈顿距离之和} = 2 + 3 + 0 + 1 + 3 + 0 + 3 + 1 = 13$

(3) $h_3(N) = \text{逆序值之和} = n_5 + n_8 + n_4 + n_2 + n_1 + n_7 + n_3 + n_6 = 4 + 6 + 3 + 1 + 0 + 2 + 0 + 0 = 16$

知识点 5.2 (启发式函数的性质)

通常启发式函数 $h(N)$ 由当前状态和目标状态二者联合确定, $h(N)$ 满足性质如下:

- (1) $h(N) \geq 0$
- (2) $h(\text{Goal}) = 0$
- (3) $h(N)$ 越小, N 离目标状态 Goal 越近



不同的启发式函数, 对搜索效率的影响不一样。

例题 5.3 (8 数码问题的启发式函数的应用) 分别利用例题 5.2 求解中设计的启发式函数 $h_1(N)$ 和 $h_2(N)$, 给出贪婪搜索算法和 A^* 算法求解图 5.9 (a) 所示 8 数码问题的过程。图 5.9 (a) 中左边节点为初始状态, 右边节点为目标状态。

解 在所有状态下方标出启发式函数 $h_1(N)$ 的值, 如图 5.9 所示。

- (1) 贪婪搜索算法 + $h_1(N)$ 求解 8 数码问题, 其过程如图 5.9 所示。
- (2) A^* 算法 + $h_1(N)$ 求解 8 数码问题, 其过程如图 5.10 所示。
- (3) 贪婪搜索算法 + $h_2(N)$ 求解 8 数码问题, 其过程如图 5.11 所示。

由上述三种情况能看出, 在都采用 h_1 的条件下, A^* 算法访问的节点数比贪婪搜索算法访问的少; 在同时都采用贪婪算法的条件下, 启发式函数选择 h_2 的算法比启发式函数选择 h_1 访问更少的节点。通常, 我们并不能说 A^* 算法比贪婪搜索更好, 这和具体的搜索问题实例相关; 但是不同的启发式函数之间有好有差, 8 数码问题中 h_2 比 h_1 好, 因为 h_2 会导致访问更少的状态。

接下来, 我们先将启发式函数进行分类, 然后以此为基础, 讨论启发式函数设计的要求。

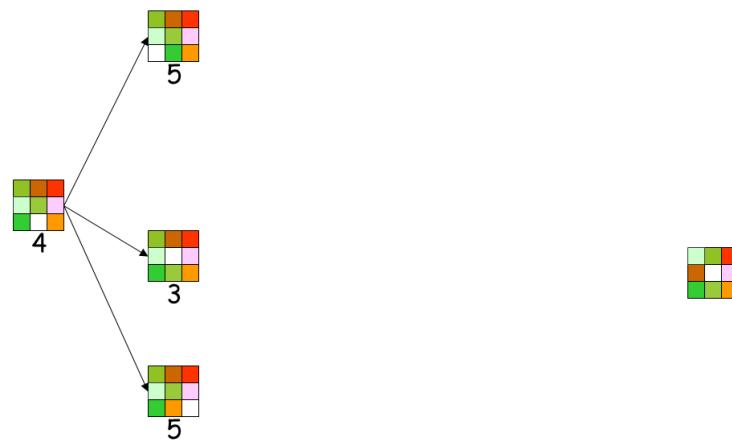
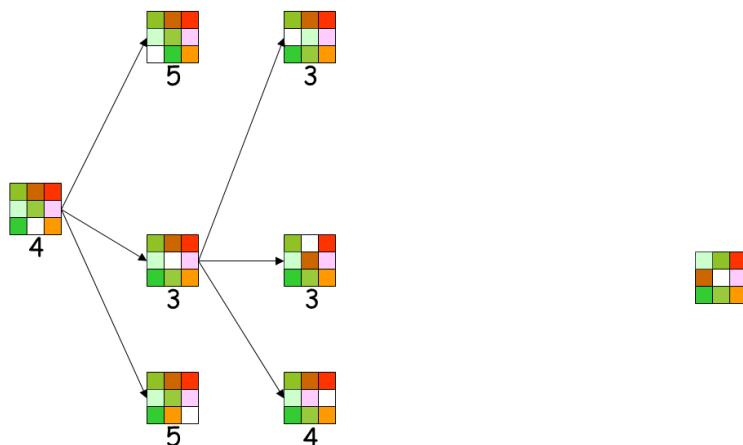
定义 5.5 (可采纳的启发式函数)

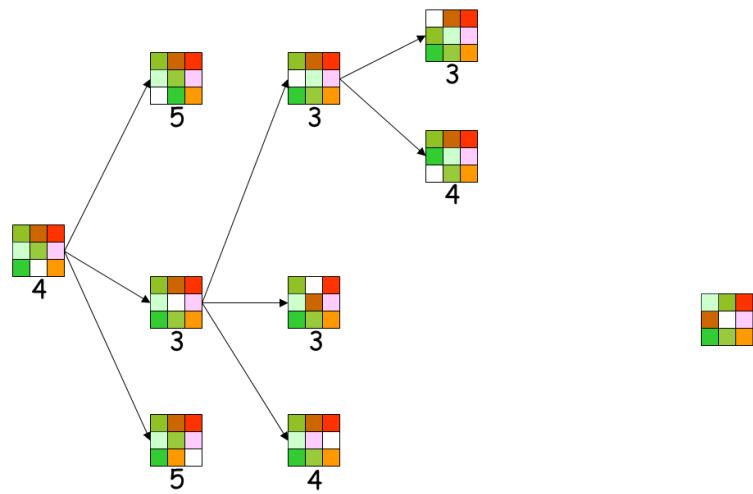
假设 $h^*(N)$ 是节点 N 到目标状态节点的实际最优路径耗散, 则启发式函数 $h(N)$ 是可采纳的, 当且仅当 $1 \leq h(N) \leq h^*(N)$ 。



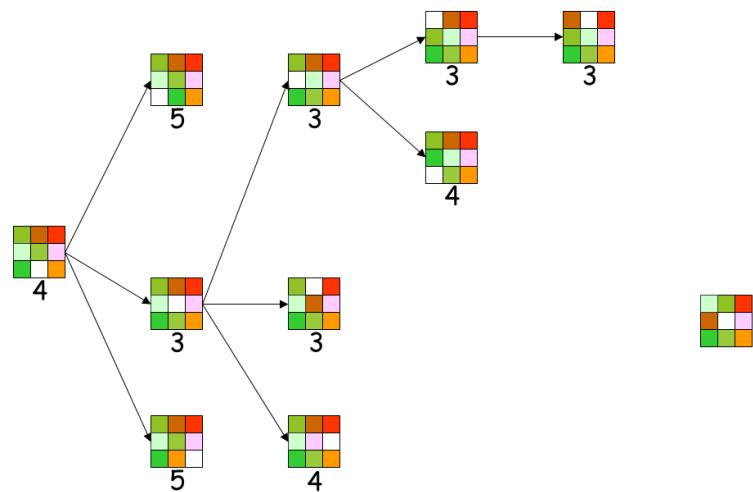
可采纳的启发式函数实现了对节点 N 到目标状态节点的路径耗散的“乐观”估计, 所谓乐观, 即对代价过低估计, 不超过实际要求的最小值。

5.4 A* 算法

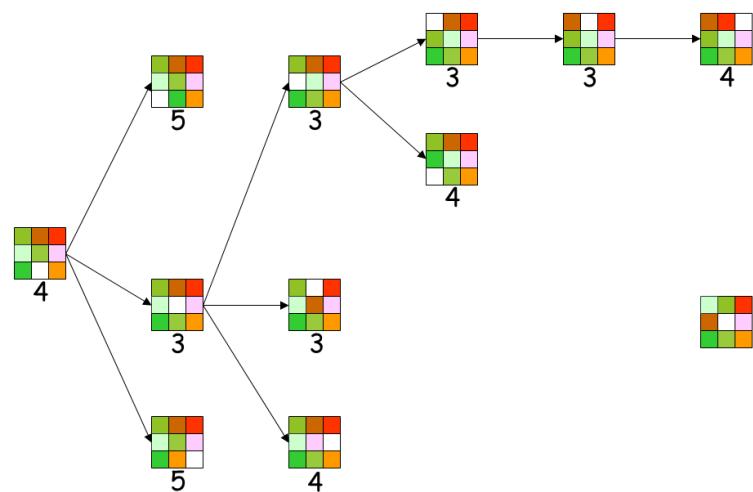
(a) 初始状态到目标状态的路径耗散估计值: $f(N) = h_1(N) = 4$ (b) 初态三个后继状态的 f 值分别为 5,3,4(c) 选择最小 f 值为 3 的状态进行扩展，并计算其后继的 f 值图 5.9: 贪婪搜索算法 $+h_1(N)$ 求解 8 数码问题



(d) 选择最小 f 值为 3 的未扩展节点，进行扩展

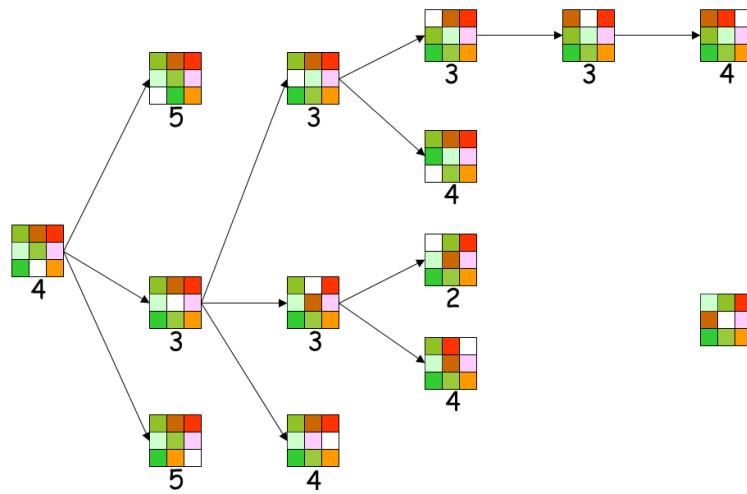
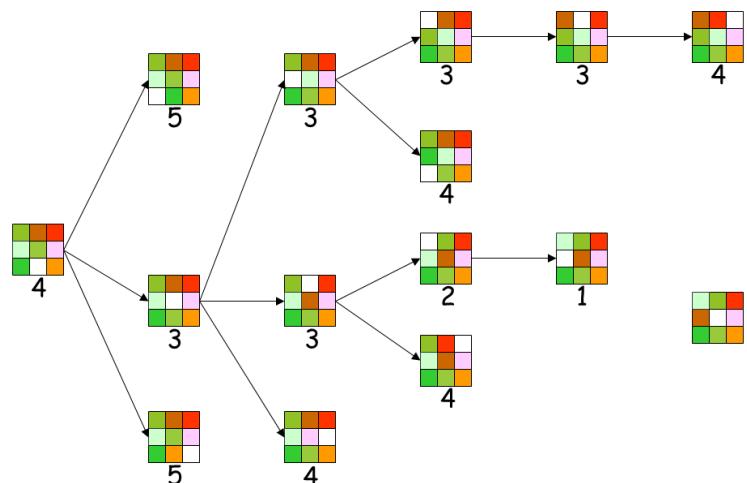
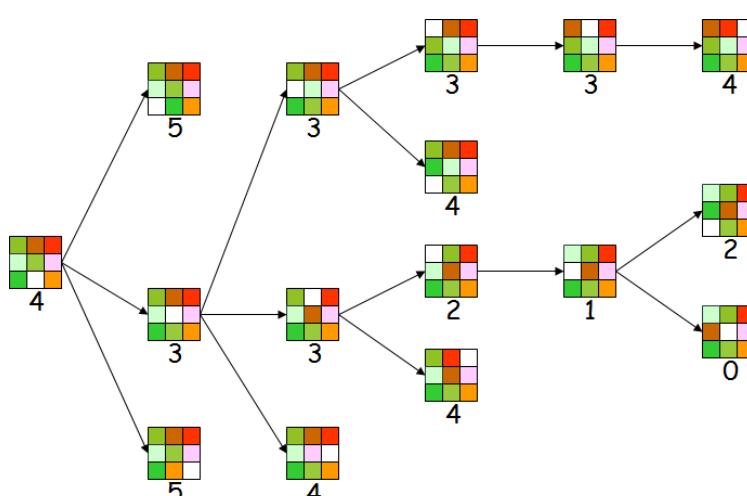


(e) 选择最小 f 值为 3 的未扩展节点，进行扩展



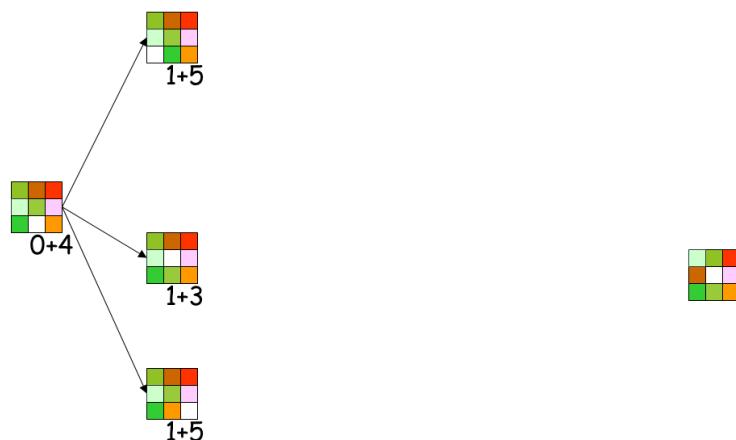
(f) 选择最小 f 值为 3 的未扩展节点，进行扩展

图 5.9: 贪婪搜索算法 + $h_1(N)$ 求解 8 数码问题 (续 1)

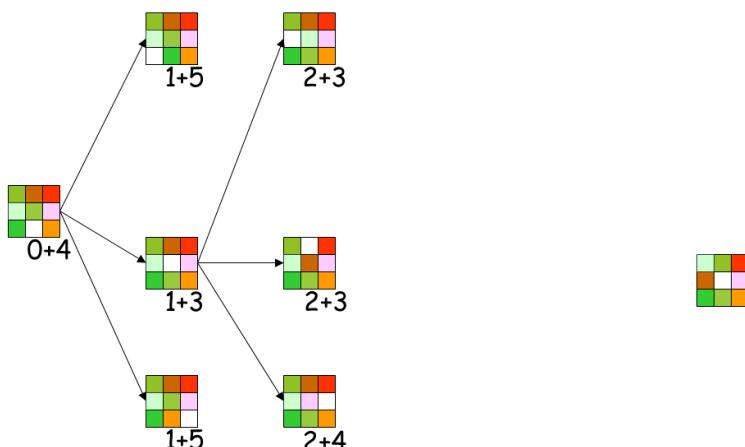
(g) 选择最小 f 值为 3 的未扩展节点，进行扩展(h) 选择最小 f 值为 2 的未扩展节点，进行扩展(i) 选择最小 f 值为 1 的未扩展节点，进行扩展，找到目标状态图 5.9: 贪婪搜索算法 + $h_1(N)$ 求解 8 数码问题 (续 2)



(a) 状态的评估值 f 写在状态节点下方，格式为“当前路径耗散 + h_1 的值”

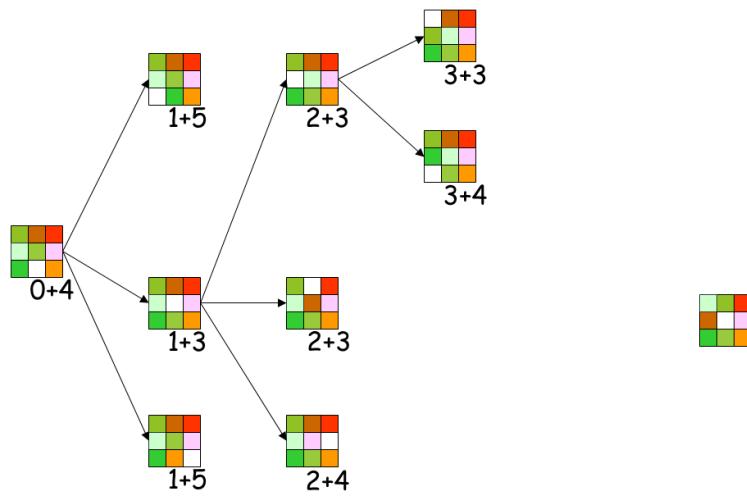
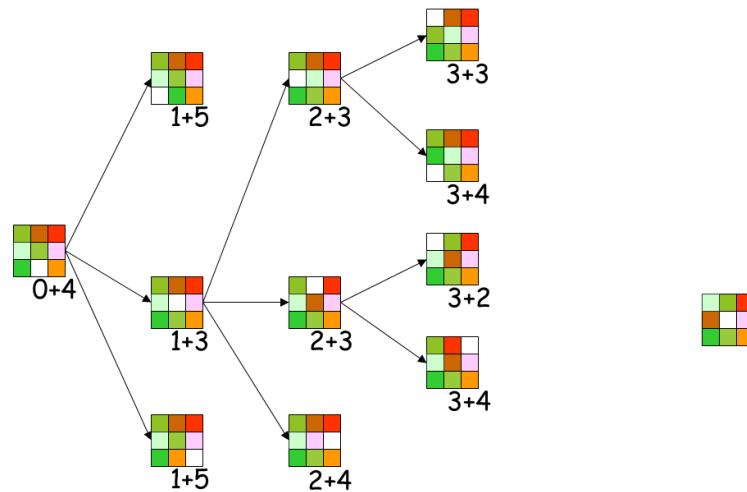
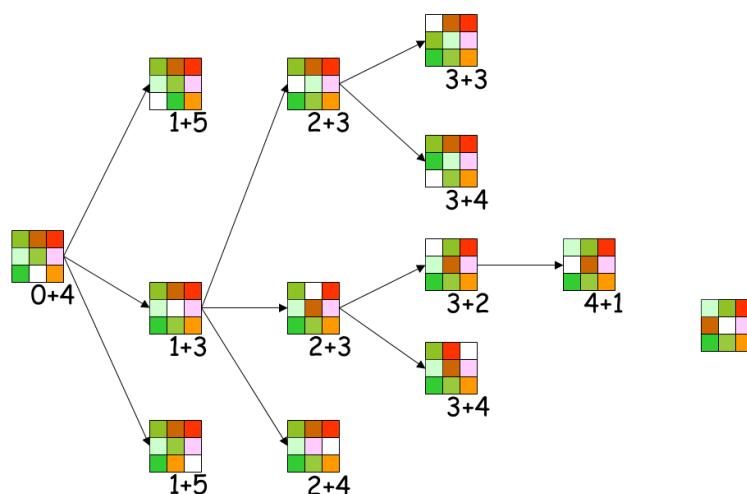


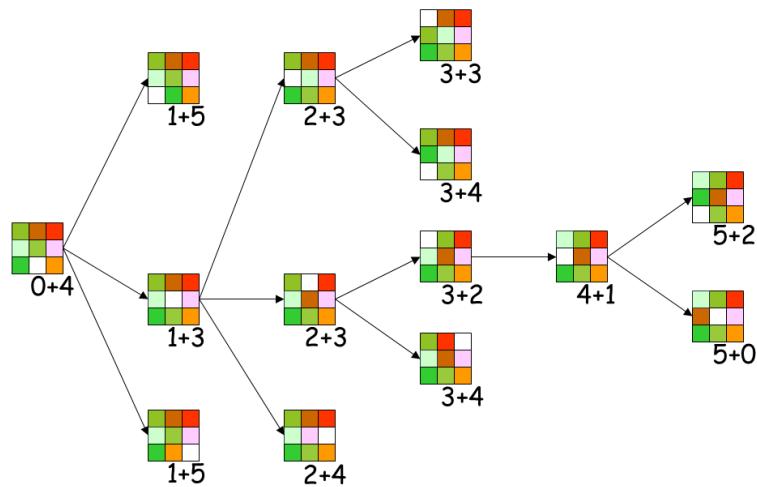
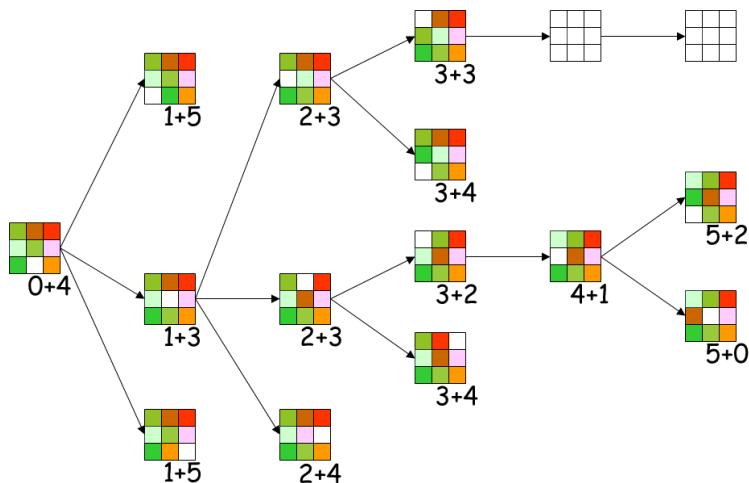
(b) 选择最小 f 值为 $0 + 4$ 的未扩展节点，进行扩展



(c) 选择最小 f 值为 $1 + 3$ 的未扩展节点，进行扩展

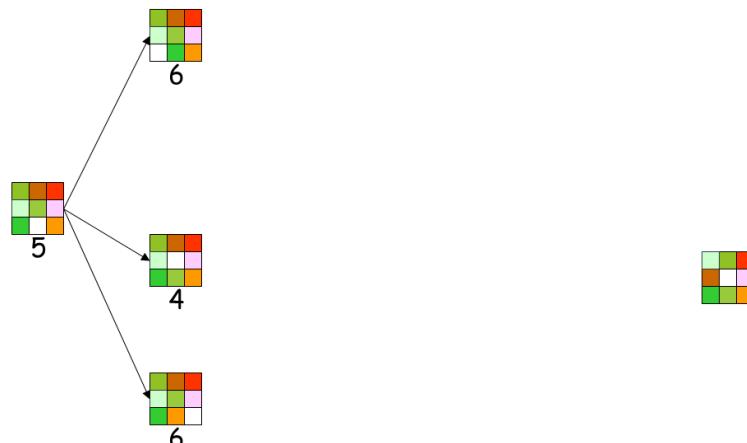
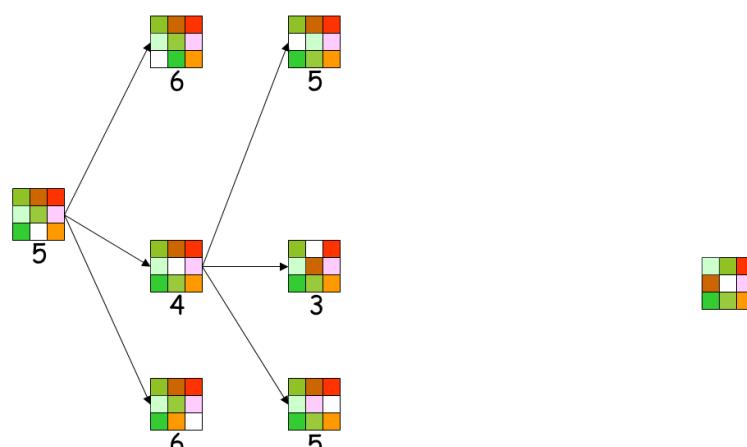
图 5.10: A^* 算法 $+h_1(N)$ 求解 8 数码问题

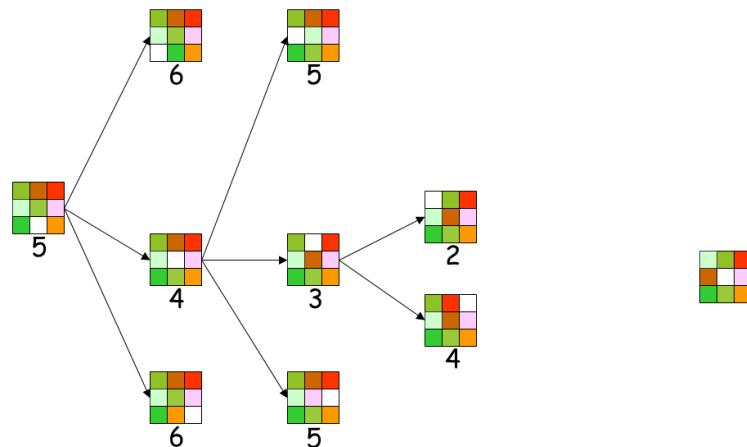
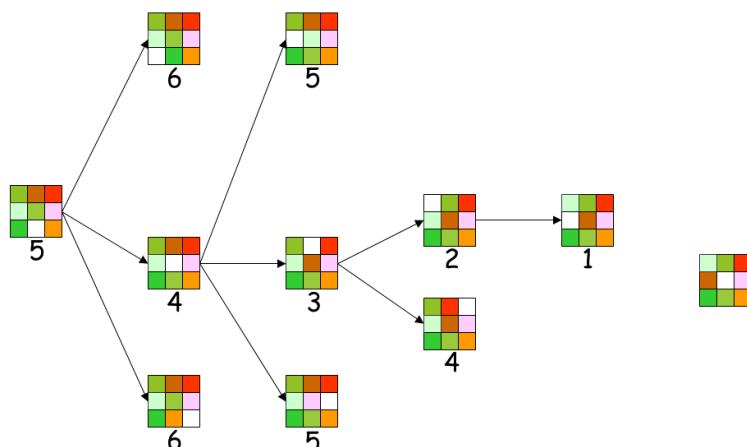
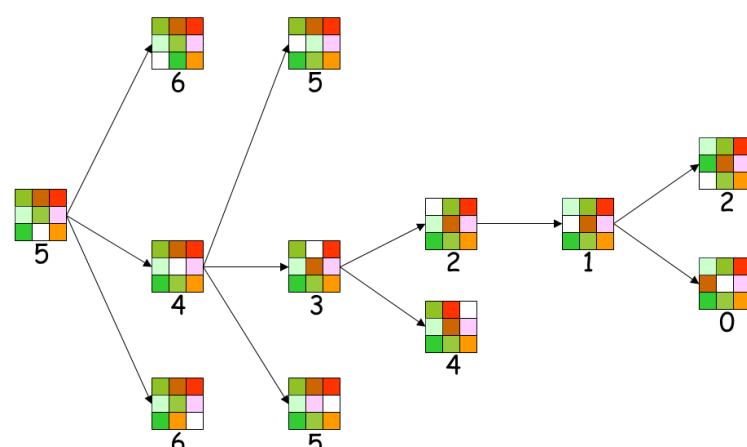
(d) 选择最小 f 值为 $2 + 3$ 的未扩展节点，进行扩展(e) 选择最小 f 值为 $2 + 3$ 的未扩展节点，进行扩展(f) 选择最小 f 值为 $3 + 2$ 的未扩展节点，进行扩展图 5.10: A^* 算法 + $h_1(N)$ 求解 8 数码问题 (续 1)

(g) 选择最小 f 值为 $4 + 1$ 的未扩展节点，进行扩展

(h) 找到目标状态，比贪婪算法少两个状态的访问

图 5.10: A^* 算法 $+h_1(N)$ 求解 8 数码问题 (续 2)

(a) 状态的评估值 $f = h_2$ 写在状态节点下方(b) 选择最小 f 值为 5 的未扩展节点，进行扩展(c) 选择最小 f 值为 4 的未扩展节点，进行扩展图 5.11: 贪婪搜索算法 + $h_2(N)$ 求解 8 数码问题

(d) 选择最小 f 值为 3 的未扩展节点，进行扩展(e) 选择最小 f 值为 2 的未扩展节点，进行扩展(f) 选择最小 f 值为 1 的未扩展节点，进行扩展图 5.11: 贪婪搜索算法 $+h_2(N)$ 求解 8 数码问题 (续 1)

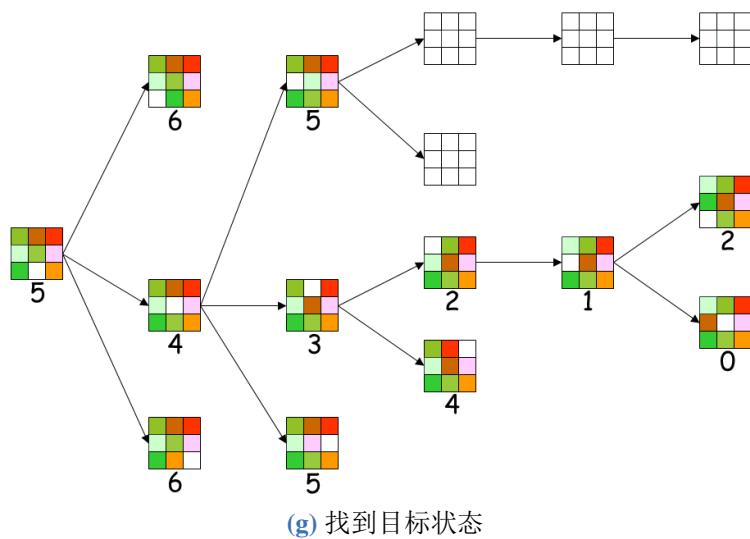


图 5.11: 贪婪搜索算法 + $h_2(N)$ 求解 8 数码问题 (续 2)

第五章 习题

1.

第六章 优化问题求解

内容提要

- 优化问题的例子
- 求解优化问题
- 优化问题搜索形式化
- 优化问题的应用

6.1 优化问题的例子

例 6.1 (连续可导函数的最优化) 求定义在闭区间 $[-1,1]$ 上的一元二次函数 $f(x) = 3x^2 - 5x$ 的最大值。

解 令 $f'(x) = 6x - 5 = 0$, 解之得到极值点 $x_0 = \frac{5}{6}$, 连同两个边界点 $x_{1,2} = \pm 1$, 得到 $f(x_0 = \frac{5}{6}) = \frac{-25}{12}$, $f(x_1 = 1) = -2$, $f(x_2 = -1) = 8$, 故, 得到最大值为 $f(x_2 = -1) = 8$ 。

优化问题求解简单说就是求一个给定函数的最值: 最大值或者最小值。当函数连续可导时, 通过求导、计算极值点, 可以快速获得函数的最值。

例 6.2 (旅行商问题/*Traveling Salesman Problem, TSP*) 给定 n 个城市以及它们两两之间的距离, 一个商品推销员要去这 n 个城市推销商品, 该推销员从一个城市出发, 需要经过所有城市恰好一次, 回到出发地; 求最短回路。

6.2 优化问题搜索形式化

6.3 求解优化问题

6.4 优化问题的应用

第六章 习题

1.

第七章 约束满足问题

内容提要

- 地图着色问题
- 约束满足问题的搜索形式化
- 定义约束满足问题
- 求解约束满足问题

7.1 地图着色问题

例 7.1 (地图着色问题) 能否用 k 种色彩对地图着色, 一个国家一种颜色, 相邻国家颜色不一样, 如图7.1所示。

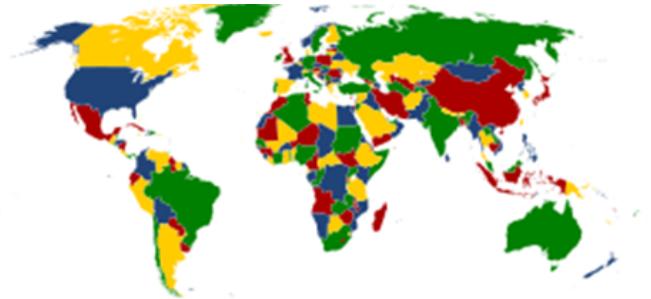


图 7.1: 四色问题

当 $k = 4$ 时, 我们有四色定理, 见知识点7.1。

知识点 7.1(四色定理)

也称“四色猜想”, 即“任何一张地图只用四种颜色就能使具有共同边界的国家着上不同的颜色”。

四色定理世界近代三大数学难题之一, 对四色定理的研究促进了图论和机器证明的发展。四色定理是第一个用计算机证明的数学定理, 证明的正文 300 页, 但是这个证明并不让所有人信服, “是否存在四色定理的一个简短证明, ……使得一个合格的数学家能在 (比如说) 两个星期里验证其正确性呢?”(摘自汤米·R·延森和比雅尼·托夫特《图染色问题》)。

地图着色问题看作搜索问题时, 将任何着色的地图视为初始状态, 每给一个国家着上满足要求的一种颜色就得到下一个后继状态, 着色问题的解不需要给出完整的路径, 不需要考虑路径耗散, 只需要给出目标状态即可。“相邻国家的着色不能相同”是约束着色的条件, 让这个条件在任意一对相邻国家间都满足, 这个问题称之为“约束满足问题”。

7.2 约束满足问题

定义 7.1 (约束满足问题 (Constraint Satisfaction Problem, CSP))

给定一个三元组 $CSP = (\mathbf{X}, \mathbf{D}, \mathbf{C})$, 其中

- \mathbf{X} 是一组 n 个变量, 不妨设为 $\mathbf{X} = (x_1, x_2, \dots, x_n)$;
- $\mathbf{D} = (\mathbf{D}_1, \mathbf{D}_2, \dots, \mathbf{D}_n)$ 是值域, 定义了变量 x_i 的取值范围 $\mathbf{D}_i, i = 1, 2, \dots, n$
- $\mathbf{C} = (C_1, C_2, \dots, C_m)$ 是一组 m 个约束条件, 是 m 个计算公式, 约束着 \mathbf{X} 的各个分量之间的取值;
- 对 $\forall x_i = v_i \in \mathbf{D}_i, i = 1, 2, \dots, n$, 若所有 n 个变量都赋了值, 得到值向量 $\mathbf{v} = (v_1, v_2, \dots, v_n)$, 值向量 \mathbf{v} 可以参与约束条件 \mathbf{C} 的计算, 判断约束是否被满足。

求变量组 \mathbf{X} 的一个赋值, 使得所有的约束条件都被满足。



例 7.2 (3-SAT) 给定 n 个布尔变量 $\mathbf{X} = (x_1, x_2, \dots, x_n) \in \{\text{true}, \text{false}\}^n$, 约束组 \mathbf{C} 包括 m 个少于等于 3 个布尔变量的析取式, 或将这 m 个析取式写成如公式 7.1 的合取式。求一个赋值 (x_1, x_2, \dots, x_n) 使得合取式 7.1 为真。

$$\overbrace{(x_1 \vee x_2 \vee x_5) \wedge (\neg x_1 \vee x_3) \wedge \dots \wedge (x_3 \vee x_n)}^{m \text{ 个析取式构成一个合取式}}, \quad (7.1)$$

3-SAT 是典型的约束满足问题。每年都有 SAT 问题求解竞赛, 网站 <http://www.satcompetition.org/> 上有历年比赛的信息, 包括代码和测试问题集等。

例 7.3 (地图着色) 如图 7.2 所示地图, 每个区域用一个变量表示, 共 7 个变量: $\mathbf{X} = \{\text{WA}, \text{NT}, \text{SA}, \text{Q}, \text{NSW}, \text{V}, \text{T}\}$, 每个变量的取值范围都是 $\mathbf{D} = \{\text{red}, \text{green}, \text{blue}\}$, 约束条件是“任何两个相邻的变量色彩不一样”, 即 $\mathbf{C} = \{\text{WA} \neq \text{NT}, \text{WA} \neq \text{SA}, \text{NT} \neq \text{SA}, \text{NT} \neq \text{Q}, \text{SA} \neq \text{Q}, \text{SA} \neq \text{NSW}, \text{SA} \neq \text{V}, \text{Q} \neq \text{NSW}, \text{NSW} \neq \text{V}\}$ 。

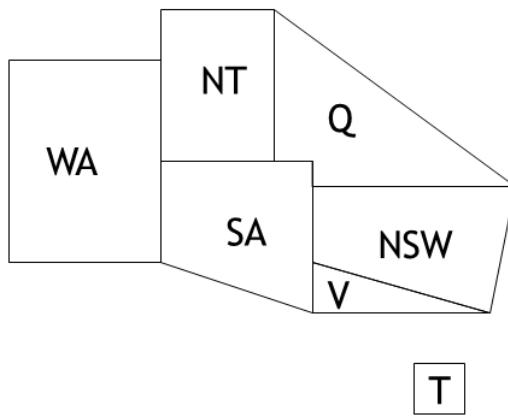


图 7.2: 地图着色例子

例 7.4 (八皇后问题) 如图 7.3 所示棋盘, 每列用一个变量表示, 共 8 个变量: $\mathbf{X} = \{x_i, i = 1, 2, \dots, 8\}$, 每个变量的取值范围都是 $\mathbf{D} = \{1, 2, \dots, 8\}$, 约束条件 \mathbf{C} 包括两类:

- (1) 若 $x_i = k$, 则 $x_j \neq k, \forall j = 1, 2, \dots, 8, j \neq i$, 即“第 k 行只能放一个皇后”;
- (2) 任意两个皇后 $\forall i, j, i \neq j$, 满足约束 $\frac{x_i - x_j}{i - j} \neq \pm 1$, 即“每个对角线只能放置一个皇后”。

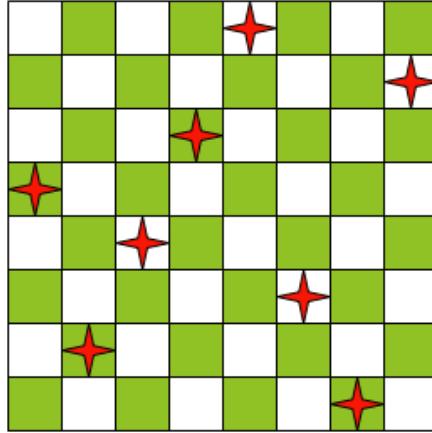


图 7.3: 八皇后问题

例 7.5 (推理问题) 假设有 5 个人，每个人有 5 种属性，一种属性视为一个属性变量，属性变量 X 和取值范围 D 如下：

- (1) 国籍 $N_i \in \{\text{英国, 西班牙, 日本, 意大利, 挪威}\}$
- (2) 住房颜色 $C_i \in \{\text{红色, 绿色, 白色, 黄色, 蓝色}\}$
- (3) 喜爱喝的饮品 $D_i \in \{\text{茶, 咖啡, 牛奶, 果汁, 水}\}$
- (4) 职业 $J_i \in \{\text{画家, 雕刻家, 外交家, 小提琴手, 医生}\}$
- (5) 养的宠物 $A_i \in \{\text{狗, 蜗牛, 狐狸, 马, 斑马}\}$

上述变量满足的约束条件 C 如下：

(1) 英国人住在红房子里。	$N_i = \text{英国} \rightarrow C_i = \text{红色}$
(2) 西班牙人养了条狗。	$N_i = \text{西班牙} \rightarrow A_i = \text{狗}$
(3) 日本人是个画家。	$N_i = \text{日本} \rightarrow J_i = \text{画家}$
(4) 意大利人爱喝茶。	$N_i = \text{意大利} \rightarrow D_i = \text{茶}$
(5) 挪威人住宅左边的第一个房子里。	$i = 1 \rightarrow N_i = \text{挪威}$
(6) 绿房子的主人爱喝咖啡。	$C_i = \text{绿色} \rightarrow D_i = \text{咖啡}$
(7) 绿房子在白房子的右边。	$C_i = \text{白色} \rightarrow C_{i+1} = \text{绿色}$
(8) 雕刻家养了蜗牛。	$J_i = \text{雕刻家} \rightarrow A_i = \text{蜗牛}$
(9) 中间房子的主人爱喝牛奶。	$i = 3 \rightarrow D_i = \text{牛奶}$
(10) 外交家住在黄色房子里。	$J_i = \text{外交家} \rightarrow C_i = \text{黄色}$
(11) 挪威人住在蓝房子隔壁。	$N_i = \text{挪威} \rightarrow C_{i+1} = \text{蓝色或 } C_{i-1} = \text{蓝色}$
(12) 小提琴手喜欢喝果汁。	$J_i = \text{小提琴手} \rightarrow D_i = \text{果汁}$
(13) 狐狸的主人住在医生的隔壁。	$A_i = \text{狐狸} \rightarrow J_{i-1} = \text{医生或 } J_{i+1} = \text{医生}$
(14) 马的主人住在外交家的隔壁。	$A_i = \text{马} \rightarrow J_{i-1} = \text{外交家或 } J_{i+1} = \text{外交家}$

请进行推理，指出每个人的五种属性分别是什么。

例 7.5 中的约束条件可以被写成约束条件右边的公式，公式中的运算符 “ \rightarrow ” 表示“蕴含”，例如 $A \rightarrow B$ 可解读为“如果 A 成立，那么 B 成立”。

知识点 7.2 (约束满足问题的分类)

依据候选解的数目是否是有限来分类约束满足问题。

- (1) 无限约束满足问题：存在至少一个变量的取值范围是无限的；
- (2) 有限约束满足问题：每个变量的可能取值个数都是有限的。

**7.2.1 约束问题的图描述**

约束满足问题可以用图来描述，由两种描述约束满足问题的图。

定义 7.2 (约束图：约束满足问题的同构图表示)

给定约束满足问题 (X, D, C) ，构造约束图 $G = \{V, E\}$ ，其中 $V = X$, $E = \{e_{ij} | x_i, x_j \text{ 出现在某个约束 } C_k \text{ 中}\}$ 。



例 7.6 (地图着色问题的约束图) 图7.4就是例7.3中的地图着色问题被描述而成的约束图。

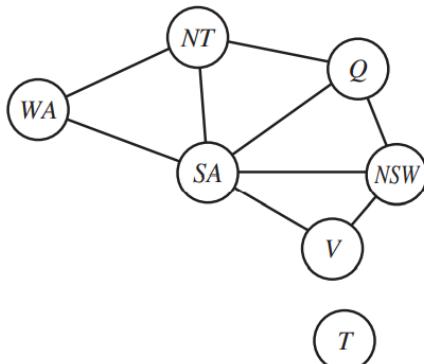


图 7.4: 地图着色问题的约束图

约束图用来描述所有约束条件只含两个变量的二元约束时，是非常简洁的。对于多元约束，可以采用因子图。

定义 7.3 (因子图：约束满足问题的异构图表示)

给定约束满足问题 (X, D, C) ，构造因子图 $G = \{V, E\}$ ，其中 $V = X \cup C$, $E = \{e_{ij} | x_i \text{ 出现在约束条件 } C_j\}$ 。



例 7.7 (地图着色问题的因子图) 如图7.5所示，将例7.3中的地图着色问题描述成因子图。

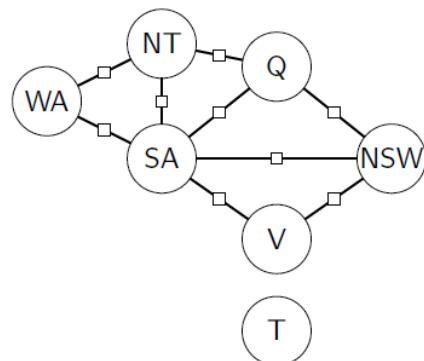


图 7.5: 地图着色问题的因子图

因子图是一种异构图，包含两类顶点：变量顶点和约束顶点。图7.5中，变量顶点用圆圈表示，约束顶点用方格表示，约束顶点也称为“因子”。图7.5的一个特点是所有因子都是度为2，表示两个相邻变量顶点不能取相同的值，即不能着同色。

习题2可以理解为因子图可以转换为约束图。

7.3 约束满足问题的搜索形式化

将约束满足问题形式化为搜索问题，需要基于定义7.4的“有效赋值”和定义7.5的“完全赋值”两个概念。为解释这两个概念，先引入约束条件状态。约束条件在求解过程中存在三种可能的状态：被满足，被违背和无法判定。若一个约束条件相关的变量都被赋值，那么约束条件是“被满足”还是“被违背”，通常都能算出结果；若约束条件相关的变量没有全部被赋值，一般情况下，该约束条件处于“无法判定”状态，无法判定并不代表着约束条件被违背或被满足。

定义 7.4 (有效赋值)

对约束满足问题中 n 个变量 x_1, x_2, \dots, x_n 中的一部分变量进行了赋值，不妨设对前 k 个变量进行了赋值，即 $x_1 \leftarrow v_1, x_2 \leftarrow v_2, \dots, x_k \leftarrow v_k, k = 0, 1, 2, \dots, n$ ，并且该部分变量赋值没有让任何约束条件被违背，该部分变量的赋值称为“有效赋值”。



定义 7.5 (完全赋值)

给定一个有效赋值，若 $k = n$ ，则该有效赋值为完全赋值，即 n 个变量都被赋值且保证所有的约束条件都被满足。



知识点 7.3 (约束满足问题的搜索形式化)

转换约束满足问题为搜索问题的五要素，具体如下：

- (1) 状态/状态空间：有效赋值/所有可能的有效赋值；
- (2) 初始状态： $k = 0$ 时的空有效赋值；
- (3) 后继函数： $\{x_1 \leftarrow v_1, x_2 \leftarrow v_2, \dots, x_k \leftarrow v_k\} \rightarrow \{x_1 \leftarrow v_1, x_2 \leftarrow v_2, \dots, x_k \leftarrow v_k, x_{k+1} \leftarrow v_{k+1}\}$ ；
- (4) 目标测试： $k = n$ ；
- (5) 路径耗散：设单步路径耗散相同，都为1。



注 约束满足问题的解不是标准搜索问题返回的路径，仅要求返回目标状态即可。

知识点 7.4 (约束满足问题的后继函数)

假设当前 n 个变量中已经有 k 个变量已经赋值，约束满足问题的后继函数有两个组成部分：

- (1) 从剩下 $n - k$ 个未赋值变量中选择一个变量，准备赋值；
- (2) 从选出变量的值域中选择一个具体的赋值。



如图7.6所示，当前黄色节点已有 k 个变量已经赋值，其后继函数首先选择一个未赋值变量 $x_j, j \in \{k + 1, k + 2, \dots, n\}$ ，在图7.6中用不同颜色的子节点表示，每种颜色代表选择了一个不同的变量，而同色节点表示同一个变量选择了不同的取值。若每个变量的值域都是一样的，大小为 s ，那么当前黄色节点的后继状态数目，即分支因子 $b = s \times (n - k)$ 。

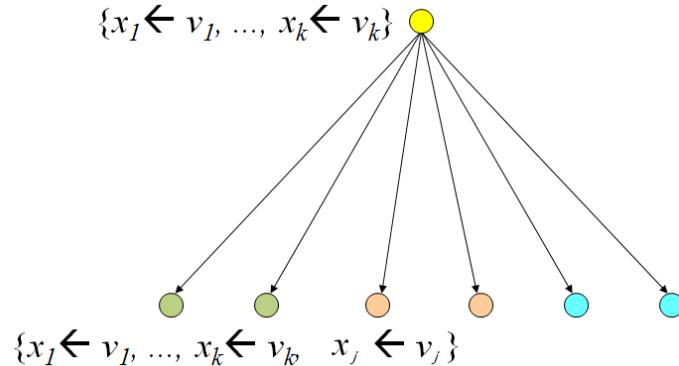


图 7.6: 约束满足问题的后继函数

知识点 7.5 (约束满足问题的性质：可交换性)

约束满足问题中变量赋值次序和完全赋值的可达性无关。



可交换性时约束满足问题区别一般搜索问题的最大特点，为求解算法的设计带来诸多便利之处。

知识点 7.6 (约束满足问题的可交换性的利用)

- (1) 不需要存储到达当前节点（已有有效赋值）的路径，可采用回溯法求解，降低内存需求；
- (2) 扩展节点的时候可以任意选择一个未赋值变量进行扩展，将分支因子由 $s \times (n - k)$ 降到 s 。



7.4 求解约束满足问题

7.4.1 回溯法

求解约束满足问题最常用的算法是回溯法，如算法7.1所示。

Algorithm 7.1: 求解约束满足问题的回溯算法(递归算法)

```

1 Function CSP-BACKTRACKING( $A$ )
Input: 变量集合  $X$ ; 约束条件  $C$ ; 有效赋值  $A$ 
Output:  $A$ : 完全赋值

2 if 有效赋值  $A$  是完全的 then
3   return  $A$ 

4  $x \leftarrow$  从变量集合  $X$  中选择一个未赋值变量
5  $D_x \leftarrow$  选择一个  $x$  所有可能取值的次序, 例如随机次序
6 foreach  $v$  in  $D_x$  do
7   Add( $x \leftarrow v$ ) to  $A$ 
8   if  $A$  是有效的 then
9     result  $\leftarrow$  CSP-BACKTRACKING( $A$ )                                /* 递归调用 */
10    if  $result \neq failure$  then
11      return  $result$ 
12   Remove( $x \leftarrow v$ ) from  $A$ 
13 return  $failure$ 
```

算法7.1通过调用 $CSP - BACKTRACKING(\{\})$ 启动执行, 即初始时刻的状态/有效赋值是 $\{\}$ 。算法7.1运行过程中减少回溯, 提高效率最关键之处在第4, 5两行的两个“选择”策略。尽管从第6行开始是对变量 x 的值域 D_x 中所有的值进行循环, 但是循环的结束有可能是通过第11行从循环体内直接返回, 因此, 第5行的 x 的次序, 对于算法的最终循环次数至关重要; 同理, 第4行需要选择一个未赋值变量 x , 尽管所有的变量都要被选择至少一次, 但是这个变量选择次序和变量的赋值会影响递归调用次数和循环次数。

讲述选择未赋值变量和变量赋值次序的策略之前, 先给出“冲突”的定义。

定义 7.6 (约束满足问题的“冲突”)

在约束满足问题求解过程中, 若变量赋值会造成约束组中至少一个约束被违背, 那么我们称这些变量的赋值相互间有“冲突”。



一旦回溯法执行过程中出现冲突, 回溯法就会取消一些变量的赋值, 执行“回溯”, 回溯就意味着算法回溯前刚访问了一些无效状态, 执行力无用的递归调用和循环, 降低了算法的效率。理想状况下, 回溯法从空集表示的有效赋值开始, 每次给变量赋值时都正确抉择了, 那么对于 n 个变量的约束满足问题, 执行 n 次赋值就可以获得一个解; 此时没有回溯发生, 算法高效地完成了求解。

7.4.2 改进的回溯法

现在考虑搜索过程中出现的一般情形, 假设算法执行过程中, 刚完成对第 k 个变量的赋值, 且赋值是错误的, 即未赋值的 $n - k$ 个变量无论用哪一种赋值, 结合已经赋值的 k 个变量赋值, 都会产生冲突, 那么能否选择出一个或尽可能少的变量予以赋值, 并基于此非完全赋值判断出第 k 个变量的赋

值是错误的，从而实现无用的递归调用和循环次数减少，搜索效率提高的目标。

前向检验技术由此被提出。

知识点 7.7 (前向检验技术)

Algorithm 7.2: 前向检验技术

```

1 Function forward-checking( $D_{\bar{A}}, x, v, A$ )
2   Input: 约束条件  $C$ ; 有效赋值  $A$ 
3   Output: 更新的  $D_{\bar{A}}, A$ ;
4   2 foreach 变量  $y \notin A$  do                                     /*  $y \in \bar{A}$  */
5     3   foreach 约束  $c \in C$  do
6       4     if  $c, y$  无关 then
7         5       continue                                         /*  $y$  没有出现在约束条件  $c$  中 */
8       6       foreach  $u \in D_y$  do
9         7         if  $A \cup \{y \leftarrow u\}$  使得  $c$  不被满足 then
10        8           remove( $u, D_y$ )

```

算法7.2中，变量集合 $X = A \cup \bar{A}$, D_y 表示变量 y 的值域, $D_{\bar{A}}$ 表示未赋值变量的值域的集合。函数参数中的 x, v 用作指示算法7.2在算法7.1的第 7 行 “ $Add(x \leftarrow v)$ to A ” 执行之后调用，可以从参数列表中删除；记录下来的是为了回溯法在递归调用结束时用于“回溯”；函数 $\text{remove}(u, D_y)$ 表示从 y 的值域 D_y 中删除值 u 。

前向检验技术是通过维护未赋值变量当前值域来找到当前值域为空集的未赋值变量，实现变量还未赋值时就能检验到冲突的目的。

例 7.8 (前向检验技术: 8 皇后问题为例) 当 $x_1 = 5$ 被赋值后, 如图7.7所示, 被黑色圆点占据的格子已经不能被未赋值变量 x_2, x_3, \dots, x_8 使用, 即它们的值域发生了改变, 需要将这些格子从未赋值变量的值域中删除。

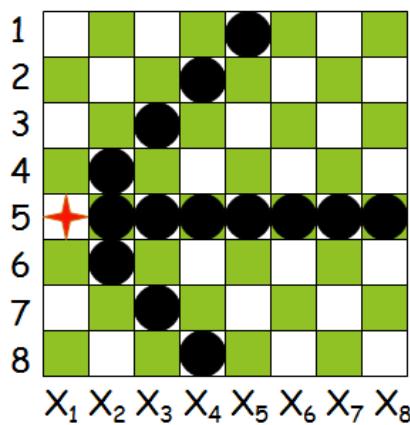


图 7.7: 前向检验技术: 8 皇后的例子

得到更新后的值域:

- $x_2 \in \{1, 2, 3, 7, 8\}$
- $x_3 \in \{1, 2, 4, 6, 8\}$
- $x_4 \in \{1, 3, 4, 6, 7\}$
- $x_5 \in \{2, 3, 4, 6, 7, 8\}$
- $x_6 \in \{1, 2, 3, 4, 6, 7, 8\}$
- $x_7 \in \{1, 2, 3, 4, 6, 7, 8\}$
- $x_8 \in \{1, 2, 3, 4, 6, 7, 8\}$

将前向检验技术加入回溯法，提高算法效率。

Algorithm 7.3: 带前向检验的回溯算法(递归算法)

```

1 Function CSP-BACKTRACKING1( $D_{\bar{A}}, A$ )
2   Input: 变量集合  $X$ ; 约束条件  $C$ ; 有效赋值  $A$ ; 未赋值变量的当前值域  $D_{\bar{A}}$ 
3   Output:  $A$ : 完全赋值
4   if 有效赋值  $A$  是完全的 then
5     return  $A$ 
6   foreach  $v$  in  $D_x$  do
7      $Add(x \leftarrow v)$  to  $A$ 
8     更新的  $D_{\bar{A}}, A \leftarrow$  forward-checking( $D_{\bar{A}}, x, v, A$ )
9     if  $D_{\bar{A}}$  的每个变量的值域非空 then
10       $result \leftarrow$  CSP-BACKTRACKING1( $D_{\bar{A}}, A$ )          /* 递归调用 */
11      if  $result \neq failure$  then
12        return  $result$ 
13      $Remove(x \leftarrow v)$  from  $A$ 
14   return  $failure$ 

```

算法7.3改进了回溯法，插入了第8行进行前向检验，修改了第9行，以前向检验的结果为条件进入递归调用。我们通过两个例子来展示算法7.3的主要思想。

例 7.9 (地图着色问题) 展示例7.3所示的地图着色问题的求解步骤。

(1) 将例7.3所示问题描述为如图7.8所示的约束图，此时所有变量的当前值域列在表7.1中。

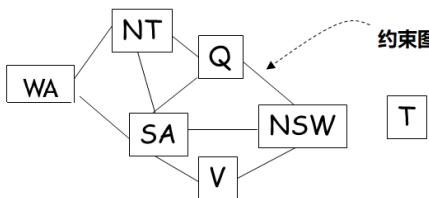
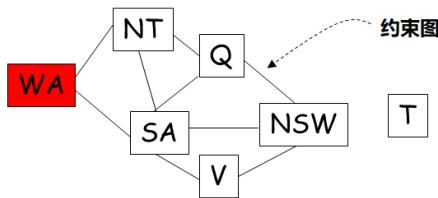


图 7.8: 地图着色问题的约束图

WA	NT	Q	NSW	V	SA	T
RGB						

表 7.1: 未赋值变量的当前值域

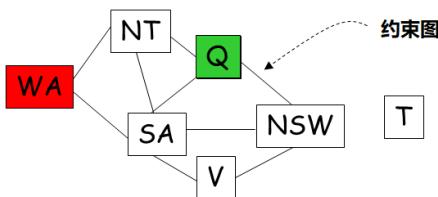
(2) 此时 $A = \{\}$ ，若选择了变量 WA，且赋值 $WA = red$ ，如图7.9所示。将 $WA = red$ 加入 A ，执行前向检验，更新未赋值变量的当前值域，也可为“残留值域”，如表7.2所示，其中 NT, SA 值域中的 red 被删除。

图 7.9: 选择 $WA = red$

WA	NT	Q	NSW	V	SA	T
RGB						
R	RGB	RGB	RGB	RGB	RGB	RGB

表 7.2: 未赋值变量的当前值域

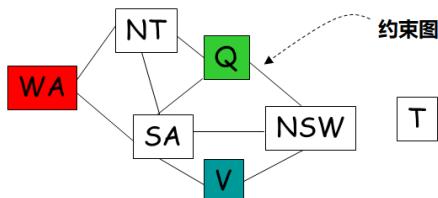
- (3) 若选择 $Q = green$, 如图7.10所示。将 $Q = green$ 加入 A , 执行前向检验, 更新未赋值变量的残留值域, 如表7.3所示, 其中 NT, SA, NSW 值域中的 $green$ 被删除, 灰色单元格表示当前的有效赋值。

图 7.10: 选择 $Q = green$

WA	NT	Q	NSW	V	SA	T
RGB						
R	GB	RGB	RGB	RGB	GB	RGB
R	GB	G	RGB	RGB	GB	RGB

表 7.3: 未赋值变量的当前值域

- (4) 若选择 $V = blue$, 如图7.11所示。将 $V = blue$ 加入 A , 执行前向检验, 更新未赋值变量的当前值域, 也可为“残留值域”, 如表7.4所示, 其中 SA, NSW 值域中的 $blue$ 被删除。此时 SA 的值域为空, 也就是说当前赋值 $\{WA \leftarrow red, Q \leftarrow green, V \leftarrow blue\}$ 不会出现在最终的解中, 因为未赋值的变量无论如何赋值都会产生冲突。因此, 需要取消 V 的赋值, 进行回溯。

图 7.11: 选择 $V = blue$

WA	NT	Q	NSW	V	SA	T
RGB						
R	B	G	RB	RGB	B	RGB
R	B	G	RB	B	B	RGB

表 7.4: 未赋值变量的当前值域

我们重新考虑步骤(4), 考虑在 $\{WA \leftarrow red, Q \leftarrow green\}$ 时, 可以考虑 $V = red/green$ 或选择其它变量赋值。选择一个变量予以赋值, 即算法7.3的第4行; 给变量选择一个最好的值, 即算法7.3的第5行。算法7.3第4, 5行选择变量和给变量赋的值, 这两个操作被称为启发式函数, 知识点7.8、7.9和7.10给出了通用的求解约束满足问题的三个常用启发式函数。

知识点 7.8 (启发式函数一: 被约束最强的变量)

选择残留值域最小的变量, 也就是被约束最强的变量。



启发式函数一倾向于选择分支因子最小的变量, 导致后继最少, 遍历后继失败后的回溯次数少。

知识点 7.9 (启发式函数二：最多被约束着的变量)

找到一个变量，赋值之后，在未满足约束组（还有变量未被赋值，所以无法判断是否满足）中出现的次数最多的，称为“最多被约束着的变量”。



启发式函数二通常在启发式函数一获得两个或多个残留值域最小的变量时启用，倾向于选择最多被约束着的变量，这个选出的变量被赋值后会参与最多的约束计算，影响其它相关的变量，可能会使得相关变量将来残留值域缩减得更多，从而降低分支因子。

知识点 7.10 (启发式函数三：最少被约束着的值)

选择值 $v \rightarrow x$ 使得未赋值的变量的残留值域中删除的值最少。



启发式函数三中的 v ，被称为“最少被约束着的值”，即这个赋值对未来的影响最小，以防将来没办法给其它变量找到赋值。

找 v 的具体做法：对每个不同的 v ，都进行前向检验，然后比较前向检验的结果，找到对残留值域大小的影响最小的 v 。

以上三个启发式函数先使用启发式函数一选择变量，若有个可选变量，再用启发式函数二选变量，若还有多个变量可选，采用随机选择策略；在给变量赋值时，按照启发式函数三赋值，具体例子如例7.10。

例 7.10 (回溯法中的启发式函数) 求解例7.3的过程，如图7.12所示。

- (1) 初始时刻，如图7.12 (a)所示，所有变量的残留值域一样，都是 $\{red, green, blue\}$ ，使用启发式函数二，选择“最多被约束着的变量”，即约束图中度最大的节点 SA，SA 被 5 个约束条件所限；SA 的取值可以任意，因为此时启发式函数三尝试所有的值，得到一样的结果；不妨令 $SA = red$ ，得到图7.12 (b)。

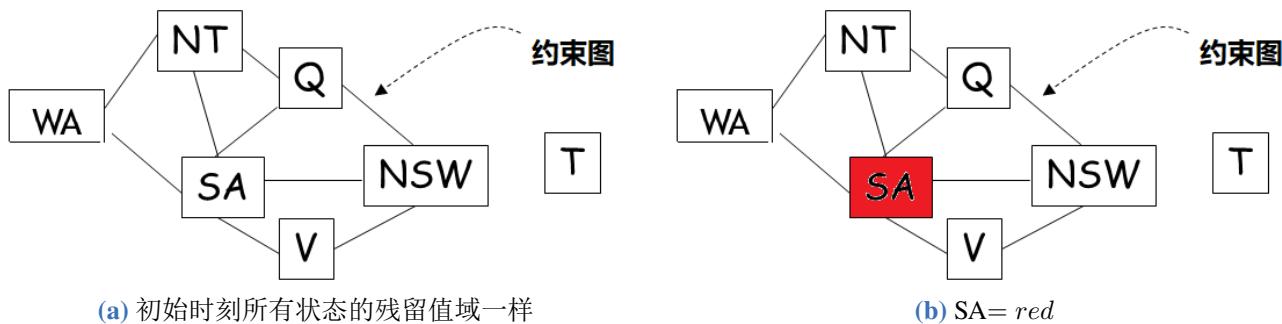


图 7.12: 回溯法求解地图着色问题

- (2) 选择第 2 个变量赋值。依据启发式函数一，WA, NT, Q, NSW 和 V 这 5 个变量的值域都是 $\{blue, green\}$ ，T 的值域是 $\{red, blue, green\}$ ，故对残留值域大小为 2 的 5 个变量使用启发式函数二，得到 NT, Q 和 NSW 赋值之后仍会出现在 2 个未满足的约束条件中，而 WA 和 V 分别出现在 1 个未满足的约束条件中；此时采用随机策略在 NT, Q 和 NSW 选择一个变量，不妨设为 NSW；再依据启发式函数三给 NSW 赋值，NSW 赋值 $blue$ 或者 $green$ ，对相关变量 Q 和 V 造成的残留值域大小一样，都是 1，所以随机给 NSW 赋值，不妨设 $NSW = green$ ，如图7.12 (c)。
- (3) 选择第 3 个变量赋值。使用启发式函数一，得到 Q 和 V 的残留值域都是 $\{blue\}$ ，而 Q 赋值后参与的未满足的约束条件有 1 个 ($Q \neq NT$)，V 赋值后参与未满足的约束条件为空，故选择变量 Q，

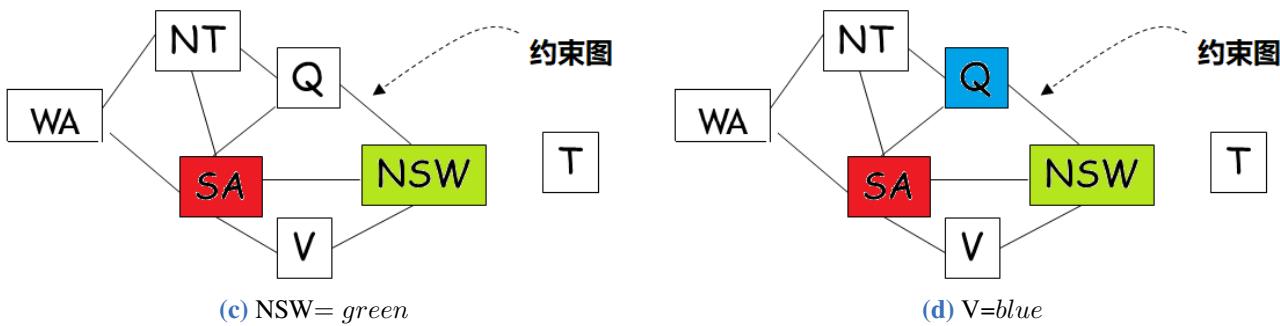


图 7.12: 回溯法求解地图着色问题(续 1)

且只能 $Q = \text{blue}$, 如图7.12 (d)。

- (4) 选择第 4 个变量赋值。使用启发法式函数一，得到残留值域最小的变量是 NT 和 V，利用启发式函数二，NT 和 V 分别赋值后，仍参与的未满足约束条件数分别为 1 和 0，故选择 NT，且只能是 $NT = green$ ，如图7.12 (e)所示。

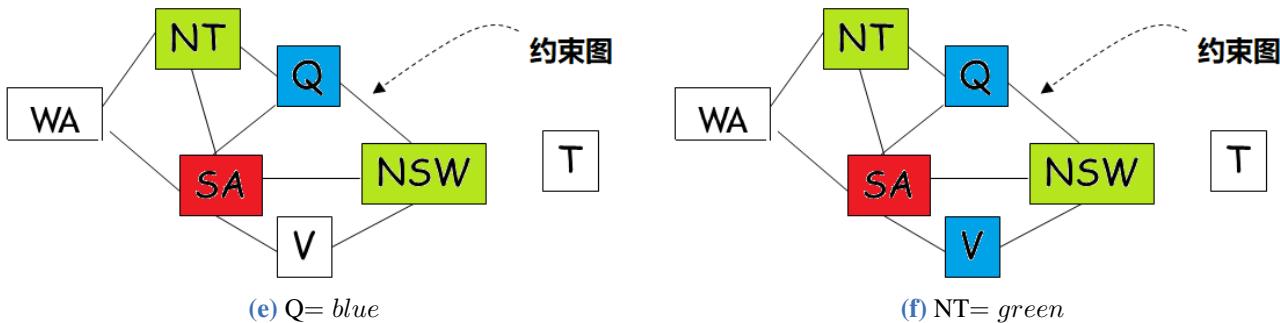


图 7.12: 回溯法求解地图着色问题(续 2)

- (5) 选择第 5 个变量赋值。使用启发式函数一，得到残留值域最小的变量是 WA 和 V；使用启发式函数二，WA 和 V 分别赋值后，仍参与的未满足约束条件数都是 0，故随机选择一个，不妨设为 V，且只能 $V = \text{blue}$ ，如图7.12 (f)所示。

(6) 选择第 6 个变量赋值。使用启发式函数一，得到残留值域最小的变量是 WA，且只能是 $WA = \text{blue}$ ，如图7.12 (g)所示。

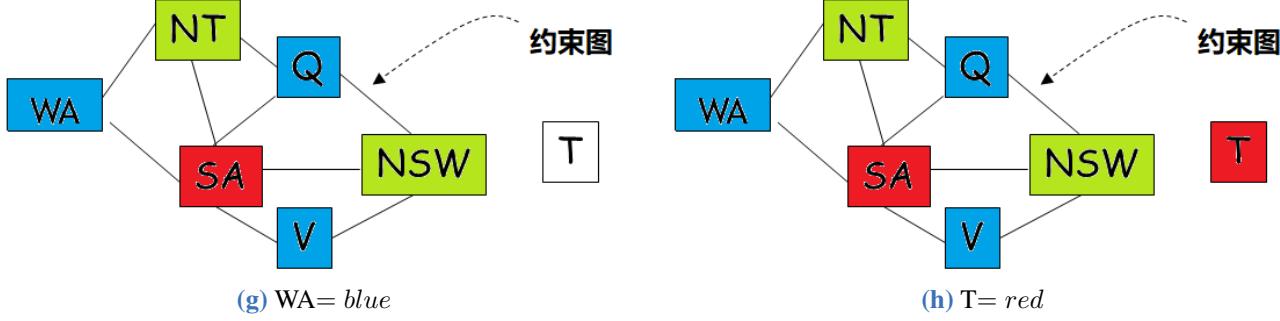


图 7.12: 回溯法求解地图着色问题 (续 3)

- (7) 选择第 7 个变量赋值, 使用启发式函数一, 残留值域最小的变量是 T; 利用启发式函数三, T 参

与了 0 个约束条件，所以它的取值可以任意设置，不妨令 $T = red$ ，如图 7.12 (h)。

我们在例 7.10 中使用了三个启发式函数，很幸运没有进行任何回溯。

知识点 7.11 (约束传播)

在前向检验完成后，若继续用迭代循环或判断邻居节点的赋值是否会导致邻居的邻居的值域为空集，这就实现了约束传播。



约束传播是前向检验技术进一步扩展。对于约束图的前向检验，只判断了当前选择出变量的直接邻居是否值域为空集；如果能找到某个变量的值域为空，相比前向检验技术，约束传播能提前“一步或多步”发现冲突。约束传播算法时间代价是传播深度的指数函数。

最后我们审视一下回溯法中的约束满足问题的状态图。

知识点 7.12 (回溯法要求的约束满足问题状态图)

一棵高度为 n 的多叉树，节点是有效赋值，其根为空集；叶节点或为完全有效赋值，或为非完全有效赋值，非完全有效赋值叶节点的扩展会违背至少一个约束条件。其中， n 是约束满足问题的变量个数。



7.4.3 局部搜索算法

不同于回溯法求解约束满足问题时定义的状态图，局部搜索算法定义的状态要求所有变量都被赋值，将约束满足问题转换为优化问题求解，优化目标设置为最小化违背的约束体条件数量。

具体局部搜索算法可以参考优化问题这一章。

采用局部搜索求解约束满足问题，有各种改进方法，例如在搜索过程中通过对经常被违背的约束条件加权，识别出“顽固且困难”的约束条件，并优先满足这些约束条件。

7.5 复杂约束满足问题的求解

当约束满足问题的变量个数和约束条件个数增加时，问题的复杂性通常指数增加，求解代价也指数增加。目前有三类思路。

寻找问题的特征，依据问题的特征设计高效的算法，受到了广泛的重视。例如约束图呈树状时，问题可以在线性时间内得到解决；而如何将一般性的约束满足问题转化为树状约束图，又成为新的关键点，但是该问题是 NP 难的，至今没有通用的有效解法。

另一个解决大规模、复杂约束满足问题的思路是利用机器学习技术，训练一个初始解生成器，生成一个“较好”的初始解，然后启动局部搜索算法。

第三个思路是结合并行计算，设计分布式约束满足问题的求解算法。这类算法的关键也还是在如何分解复杂问题为若干易于并行求解的小规模问题。

第七章 习题

1. 例 7.10 中选择第 2 个变量赋值时，若选择变量 Q ，请给出接下来的过程，并比较和例 7.10 中求解过程的差异。
2. 证明有限约束满足问题总是可以转换成只有二元约束的约束满足问题。(提示：先考虑三元约束如何通过引入辅助变量，转换多个二元约束，然后再考虑更多变量的约束。)

第三部分

高级问题求解

第八章 马尔科夫决策过程

内容提要

- 搜索问题的变型
- 最优策略
- 定义马尔可夫决策过程
- 策略评估

8.1 搜索问题的变型

搜索问题中，状态图的边参与描述后继函数，也可以被视为是边两端的顶点描述的状态发生了“转移”，这个转移可以认为是智能体做出了一个决策或行动之后发生的。这一章讨论智能体做出一个动作之后，智能体的状态会沿着边转移，当前状态与其它状态的连边表示“可能的”状态转移方向。这里的“可能的”表示在动作发生后，状态沿哪一条边转移，方向是“不确定的”。这是对经典搜索问题的一种扩展，能适应更多实际问题。例如：抛硬币游戏，当前硬币正面朝上，“抛硬币”是个动作，动作执行后，硬币落下静止时状态可能是正面朝上，也可能是背面朝上，是不确定的，可能受到抛硬币的力量，环境风速，抛前朝向等各种因素的影响。在问题建模的时候，经典搜索问题形式化方法常不能将所有影响状态转移的因素都考虑进去，因此需要更一般、更实用的描述方法。

经典搜索问题的状态图可以用描述所有边(顶点状态 1, 顶点状态 2, 边信息)的列表描述。我们引入智能体做动作决策引发状态转移的观点，适应更实用的场景，此时搜索问题对应的状态图的边有两种改进的描述方式。

第一种方式是(顶点状态 1, 顶点状态 2, 边信息, 动作)，每条边是一个不同的动作，一个动作只能实现从顶点状态 1 到顶点状态 2 的转移。这种方式动作种类太多，处理不方便，我们更常用第二种描述方式。

第二种方式定义 $k \leq 1$ 个动作，若 n 是状态的总数，那么任意一个状态的一个动作会转移到 n 个不同的状态，得到一个完全有向图， k 个动作得到 k 个可能不同的完全有向图，如图8.1所示 ($k = 2$)；完全有向图的边被描述成(顶点状态 1, 顶点状态 2, 边信息, 动作, 转移概率)，描述中添加了一个“转移概率”，表示从顶点状态 1 转移到顶点状态 2 的概率，若这些概率只能取值 0 或 1，那么当我们移除概率为 0 的边，即不可能发生转移的边，合并约简后的 k 个图，得到的状态图描述了一个经典搜索问题。当这些概率的取值范围扩展到 $[0, 1]$ 区间，用概率综合简化描述状态转移时那些没有被考虑进去的因素时，我们就得到新的搜索问题的变型。

例 8.1 (新搜索问题的例子) 如图8.1所示，1, 2, 3, 4 代表 4 个不同的状态，虚线和实线分别代表两个不同的动作， $p_{ij} \geq 0$ 表示“实线”动作导致状态 i 转移到状态 j 的概率，对任意 $i = 1, 2, 3, 4$ ，满足 $\sum_{j=1}^4 p_{ij} = 1$ ；类似的， $p'_{ij} \geq 0$ 表示“虚线”动作导致状态 i 转移到状态 j 的概率，对任意 $i = 1, 2, 3, 4$ ，满足 $\sum_{j=1}^4 p'_{ij} = 1$ 。

经典搜索问题行动会让搜索沿着一条“确定的”边前进，解可以描述成从初态到目标状态的一条路径，路径可记为：(初态, 边 1, 边 2, …, 边 m , 目标状态)，或者 (初态, 状态 1, 状态 2, …, 状态 m , 目标状态)，或者行动序列 (初态, 行动 1, 行动 2, …, 行动 m , 目标状态)，这三种解的描述对经典搜索问题是等价的。

而新搜索问题依然要求智能体找到从指定初态到目标状态的一条（最短）路径，但是智能体并不能控制自身从状态图的哪一条边进行状态转移，只能控制智能体做出动作。如图8.1所示，智能体只能

在当前状态下决策是做“实线”8.1(a), 还是“虚线”8.1(b)所代表的动作, 动作一旦完成, 智能体会沿哪一条边进行状态转移是随机的, 状态转移服从图中边上标识的概率分布。因此, 假设智能体不断从初态出发, 每次都采取一样的策略, 智能体每次可能会走一条不同的路径; 这种路径具有随机性, 且路径数目可能是无穷的。路径的“随机性”使得要找到从初态到目标状态的“固定”路径是困难的, 我们只能让智能体在每个状态下, 做出“正确的”决策或动作, 追求“以最大的概率”找到“最短路径”。也就是说, 求解新搜索问题不要求返回一条路径, 这样一条路径并没有意义, 因为无法控制智能体就沿该路径前进; 而是要求返回一个策略, 描述了每个状态下智能体的动作决策, 使得在该策略下找到从初态到目标状态的路径是最优的。

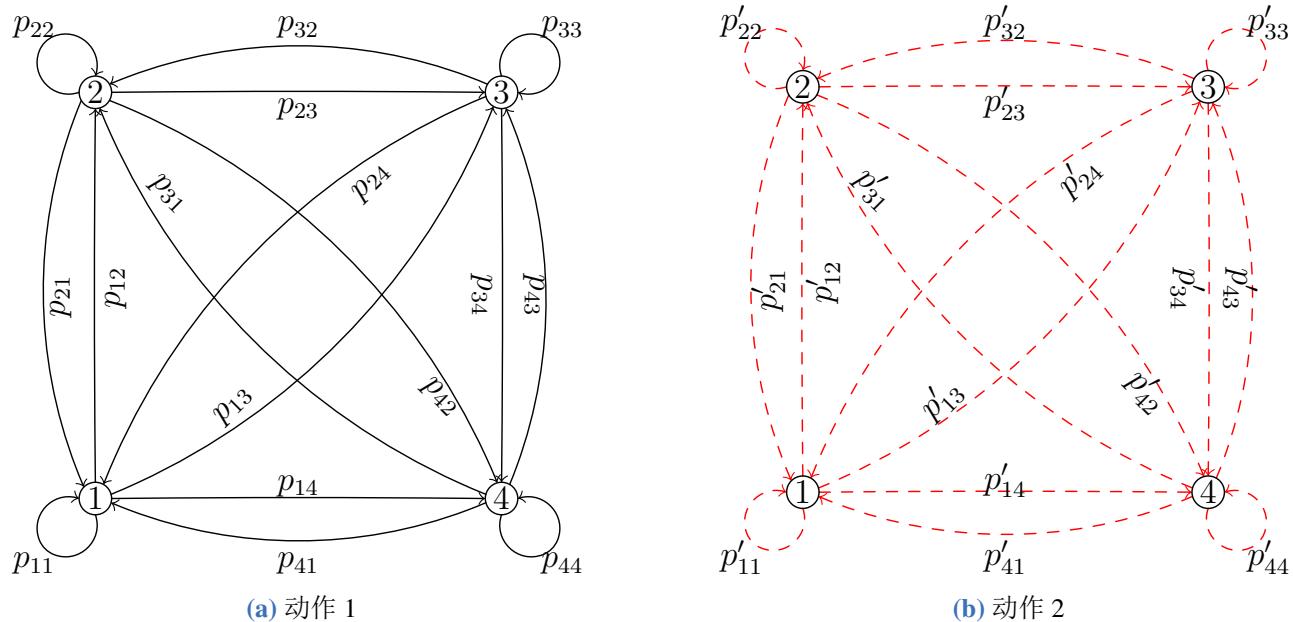


图 8.1: 新搜索问题的例子

例 8.2 (随机数游戏) 如图8.2所示, 游戏为回合制, 每个回合开始时, 有两个动作可选: 继续游戏 (黑色实线) 或者退出游戏 (红色虚线); 若选择退出游戏, 则得到¥15, 且游戏结束; 若选择继续游戏, 则得到¥4, 游戏继续; 此时产生一个 $0 \sim 9$ 的随机数, 若该随机数为 $0, 1, 2$ 之一, 则游戏直接结束; 否则, 该回合结束。为获得最多的收益, 你在游戏中时, 该如何决策, 选择“继续”还是“退出”?

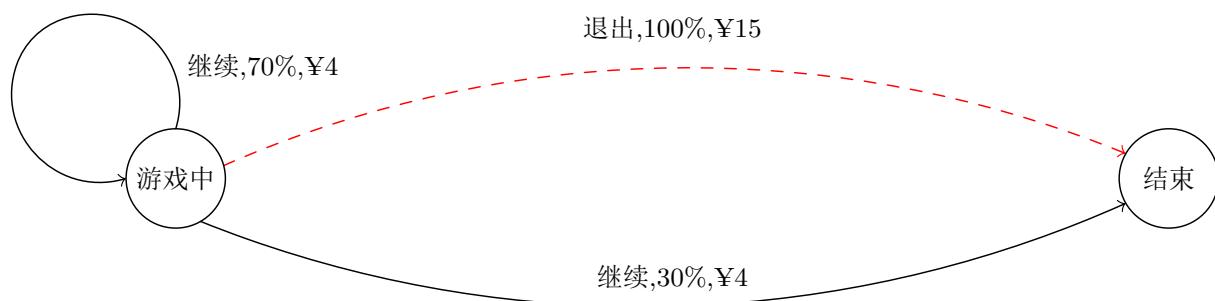


图 8.2: 随机数游戏

解 分析问题:

- (1) 若第一回合主动退出, 收益为: ¥15, 获得该收益的概率为 100%;
- (2) 若第二回合主动退出, 收益为: ¥19, 获得该收益的概率为 70%;
- (3) 若第三回合主动退出, 收益为: ¥23, 获得该收益的概率为 49%;

(4) ...

(5) 第 k 回合主动退出，收益为： $\text{¥}15 + 4k$ ，获得该收益的概率 100 为 $0.7^k \times 100\%$ 。

若每个回合都选择“继续”，获得各种收益的概率如图8.3所示，计算过程如下：

- (1) 第一回合结束后退出游戏，收益为： $\text{¥}4$ ，获得该收益的概率为 30%
- (2) 第二回合结束后退出游戏，收益为： $\text{¥}8$ ，获得该收益的概率为 21% ；
- (3) 第三回合结束后退出游戏，收益为： $\text{¥}12$ ，获得该收益的概率为 14.7% ；
- (4) ...；
- (5) 第 k 回合结束后退出游戏，收益为： $\text{¥}(4k)$ ，获得该收益的概率为 $0.7^{k-1} * 0.3 \times 100\%$ ；

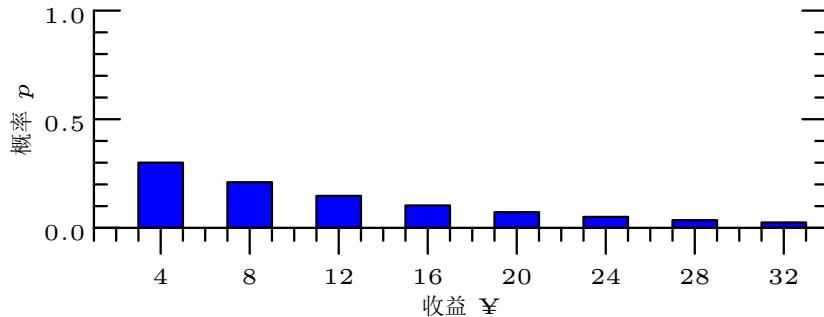


图 8.3: 随机数游戏，每回合都“继续”的收益分布

每回合都“继续”，期望的收益为 $0.3 * 4 + 0.7 * 0.3 * 8 + 0.7 * 0.7 * 0.3 * 12 + \dots = 40/3$ 。**注** 随机数游戏的状态图8.2中不同的线型达标不同的行动，概率为0的边被删除。

8.2 定义马尔可夫决策过程

接下来，我们用“马尔可夫决策过程”来描述上述新的搜索问题。

定义 8.1 (马尔可夫决策过程)

定义马尔可夫决策过程为一个六元组 $MDP = (\mathbb{S}, \mathbb{A}, s_0, T, r, Goal)$ ，其中

- \mathbb{S} 是状态空间；
- \mathbb{A} 是动作空间，也称策略空间，是所有动作/策略构成的集合；
- $s_0 \in \mathbb{S}$ 是给定的初态；
- 状态转移函数： $T : \mathbb{S} \times \mathbb{A} \times \mathbb{S} \rightarrow [0, 1]$ ， $T(s, a, s')$ 表示从状态 s 出发，采用动作 a ，转移到状态 s' 的概率，且满足 $\sum_{s' \in \mathbb{S}} T(s, a, s') = 1$ 或 $\int_{\mathbb{S}} T(s, a, s') ds' = 1$ ；通常称 s 为当前状态，称 s' 为下一状态或次态；
- 奖励函数 $r : \mathbb{S} \times \mathbb{A} \times \mathbb{S} \rightarrow \mathbb{R}$ ， $r(s, a, s')$ 表示从状态 s 出发，采用动作 a ，转移到状态 s' 获得的奖励；
- 目标测试 $Goal(s) == true$ 表示找到目标状态 s 。



图8.2中每条边代表的状态转移标记了权值“动作，概率，收益”。

与经典搜索问题比较，马尔可夫决策过程与之不同之处在于：

- (1) 动作和状态转移函数联合作用，类似于经典搜索问题的后继函数；任意给定一个状态 s 和任意一个行动 a ，其状态转移到一个可能的“后继状态”集合，而转移到这些可能后继状态的概率形成

一个分布，即 $\sum_{s' \in S} T(s, a, s') = 1$ （离散情形）。若重新定义概念“后继状态”为：若 s' 是 s 的后继状态，当且仅当 $T(s, a, s') = 1$ 。这种定义下，经典搜索问题中，对任意给定的一组状态 s 和行动 a ，有唯一后继状态 s' 或没有后继状态；

- (2) 奖励函数对应经典搜索问题的路径耗散，马尔可夫决策过程关注最大化路径奖励；
- (3) 从当前状态转移到其它状态（含当前状态）的概率分布只与当前状态以及当前采用的动作相关。

 **笔记** 马尔科夫假设。俄国统计学家安德烈·马尔科夫(1856-1922)第一个深入研究了状态转移模型，若次态只依赖于相邻的前 k 个时刻的状态，那么这个假设称为 k -阶马尔科夫假设。我们仅讨论1-阶马尔科夫假设下的问题。

例8.3 (随机数游戏转化为马尔可夫决策过程)描述随机数游戏的六元组 $MDP = ((S, A, s_0, T, R, Goal))$ 如下：

- (1) 状态空间 $S = \{\text{结束}, \text{游戏中}\}$ ；
- (2) 初态： $s_0 = \text{游戏中}$ ；
- (3) 动作空间： $A = \{\text{继续}, \text{退出}\}$ ；
- (4) 状态转移函数： $T(s, a, s')$

$$T(s, \text{继续}, s') = \begin{pmatrix} 1 & 0 \\ 0.3 & 0.7 \end{pmatrix} \quad T(s, \text{退出}, s') = \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix}$$

- (5) 奖励函数： $r(s, a, s')$

$$r(s, \text{继续}, s') = \begin{pmatrix} 0 & 0 \\ 4 & 4 \end{pmatrix} \quad r(s, \text{退出}, s') = \begin{pmatrix} 0 & 0 \\ 15 & 0 \end{pmatrix}$$

- (6) 目标测试： $Goal(\text{结束}) = true$ 。

注 上述4个矩阵中，第1行和第1列表示“结束”状态；第2行和第2列表示“游戏中”状态，行表示当前状态 s ，列表示下一状态/次态 s' 。

知识点 8.1 (马尔可夫决策过程的状态与动作)

马尔可夫决策过程的每个状态可以对应一个不同的动作集合，也可以让所有的状态对应到一个一样的动作集合，差别仅在于动作空间的大小。不妨设状态 s 对应的动作空间为 A_s ，那么我们令 $A = \bigcup_s A_s$ ，表示所有可能动作的集合，若一个动作 $a \in A$ 不在状态 s 的原本动作空间 A_s 中，即 $a \notin A_s$ ，那么可以令 $T(s, a, s) = 1[s' == s]$ ， $r(s, a, s') = 0$ 。

注 公式 $T(s, a, s) = 1[s' == s]$ 中的方括号表示“条件”运算，若条件 $s' == s$ 成立，那么 $T(s, a, s) = 1$ ，否则 $T(s, a, s) = 0$ 。

知识点 8.2 (马尔可夫决策过程的奖励)

马尔可夫决策过程的奖励函数 $r(s, a, s')$ 是动作 a 完成后状态转移到 s' 后的“实时”奖励，可以认为是对动作的奖励；但很多实际问题中奖励 $r(s, a, s')$ 只与次态 s' 相关，例如下棋，不管 s, a 是什么值，棋局分出胜负就给非0的奖励，否则给奖励0。

定义 8.2 (策略：马尔可夫决策过程的解)

给定一个马尔可夫决策过程 $MDP = (\mathbb{S}, \mathbb{A}, s_0, T, r, Goal)$, 对状态空间的每个状态 $s \in \mathbb{S}$ 定义一个动作 $a \in \mathbb{A}$, 构成一个映射表 π , π 被称为“策略”或“策略函数”。 

上述策略是“确定性的”，即每个状态只对应一个动作，100% 选择该动作执行；若策略函数输出是一个动作空间上的分布，描述选择动作空间中的每个动作的概率，那么该策略就是“非确定性的”。这里仅考虑确定性策略。

例 8.4 (随机数游戏的策略) 有如下 4 中不同的策略：

- (1) $\pi_1(\text{游戏中}) = \text{继续}, \pi_1(\text{结束}) = \text{退出};$
- (2) $\pi_2(\text{游戏中}) = \text{继续}, \pi_2(\text{结束}) = \text{继续};$
- (3) $\pi_3(\text{游戏中}) = \text{退出}, \pi_3(\text{结束}) = \text{退出};$
- (4) $\pi_4(\text{游戏中}) = \text{退出}, \pi_4(\text{结束}) = \text{继续}.$

上述随机数游戏的策略是确定性的，每个状态都只对应一个动作。这 4 个策略优劣可能各不相同，接下来讨论如何获得最优策略。

8.3 策略的价值

例 8.4 的随机数游戏可以进一步泛化，得到一般马尔可夫决策过程的策略数目为 $|\mathbb{A}|^{|\mathbb{S}|}$ 或 $\prod_s |A_s|$ ，其中 $|\mathbb{A}|$ 和 $|\mathbb{S}|$ 分别是动作空间 \mathbb{A} 和状态空间 \mathbb{S} 的大小， $|A_s|$ 是状态 s 对应的动作空间 A_s 的大小。马尔可夫决策过程的策略数目如此庞大，哪一个策略更好呢？令智能体执行策略，检视其结果，由此判定策略的优劣。

知识点 8.3 (策略的优劣)

若智能体依据策略执行搜索，则会生成一条从初态到目标状态的随机路径；这种随机路径的数目一般是状态数目的指数函数。一条随机路径的优劣并不能代表策略的优劣；策略的优劣应是该策略产生的所有可能随机路径优劣的综合。 

分析策略的优劣的判断要基于动作收益和路径收益。

一次状态转移产生的奖励为 $r(s, a, s')$, 也可以认为是给定当前状态为 s 时，采用动作 a , 转移到特定次态 s' 获得的“动作收益”，即 $U(s, a, s') = r(s, a, s')$ 。注意到 $U(s, a, s')$ 并不足以描述动作 a 的所有特点，因为该收益计算中还有两个变量 s 和 s' ，也就是说动作在不同的 s, s' 条件下，收益可能是不一样的。我们定义动作收益，去掉 s' 的影响，得到定义 8.3。

定义 8.3 (动作收益)

给定一个马尔可夫决策过程 $MDP = (\mathbb{S}, \mathbb{A}, s_0, T, r, Goal)$, 在当前状态为 $s \in \mathbb{S}$ 时，采用动作 $a \in \mathbb{A}$, 导致不同次态，计算这些不同次态下收益的期望值，称为动作收益，即 $R(s, a) = \sum_{s' \in \mathbb{S}} T(s, a, s')U(s, a, s')$, 其中 $U(s, a, s') = r(s, a, s')$. 

给定一个马尔可夫决策过程 MDP 和策略 π , 假设当前时刻为 t 时刻，状态记为 s_t , 一次状态转移的收益的期望值为 $R(s_t, \pi(s_t))$ 。但是在很多实际问题中，一个动作完成后，收益经常为 $r_t = 0$, 比如下棋，每一步结束后实时奖励都是 0，只有棋局分出胜负了，才给出非 0 的奖励。在 t 时刻如何对动作 a 做出评价？我们定义长期收益（或称为长期回报）。

定义 8.4 (长期收益)

给定一个马尔可夫决策过程 MDP 、策略 π ，和该策略下的一条随机路径 $(s_t, a_t, r_t, s_{t+1}, a_{t+1}, r_{t+1}, s_{t+2}, \dots)$ ，定义 t 时刻的积累奖励为（状态为 s 时策略 π 的）长期收益，即： $G_t = R(s_t, \pi(s_t)) + R(s_{t+1}, \pi(s_{t+1})) + \dots$



注意到定义 8.4 的长期收益是和一条给定的随机路径相关的，但在实际应用中，次态 s_{t+1} 并不是一个固定的状态，所以次态的奖励期望值 $R(s_{t+1}, \pi(s_{t+1}))$ 也许是另一个值 $R(s'_{t+1}, \pi(s'_{t+1}))$ ，即会导致 G_t 的定义和计算按照定义 8.4 就会存在不确定性。换个角度看，就是计算 t 时刻状态为 s 时策略 π 的长期收益时，公式中的次态收益并不“可靠”，因此，可以采用一个“打折”的策略来近似解决这个不可靠的次态收益。

定义 8.5 (打折的长期收益和折扣因子)

给定一个马尔可夫决策过程 MDP 和策略 π ，定义 t 时刻打折的积累奖励为（状态为 s 时策略 π 的）长期收益，即： $G_t = R(s_t, \pi(s_t)) + \lambda R(s_{t+1}, \pi(s_{t+1})) + \lambda^2 R(s_{t+2}, \pi(s_{t+2})) + \dots$ ，其中 $0 < \lambda \leq 1$ 称为折扣因子。



有了每次状态转移的动作收益，还可以定义路径收益。

定义 8.6 (路径收益)

给定一条随机路径 $s_0, a_0, s_1, a_1, \dots, s_n$ ，其中 s_0 是初态， $Goal(s_n) = true$ ，路径收益是每次状态转移的奖励之和，即 $u = r(s_0, a_0, s_1) + r(s_1, a_1, s_2) + \dots + r(s_{n-1}, a_{n-1}, s_n)$ 。



得到了每条随机路径的收益，我们可以综合这些随机路径的收益，例如给定策略 π ，其产生的所有随机路径的收益的期望值可用来评价策略。

定义 8.7 (策略收益/策略价值：定义 1)

给定一个马尔可夫决策过程 MDP ，给定策略 π ，重复执行策略 π ，获得多条随机路径为 $path_1, path_2, \dots$ ，每条随机路径的收益分别为 u_1, u_2, \dots ，每条路径出现的概率分别为 p_1, p_2, \dots ，则策略 π 的价值 $V_\pi(s) = p_1 u_1 + p_2 u_2 + \dots$



我们也可以从长期收益的定义出发定义策略的价值。

定义 8.8 (策略收益/策略价值：定义 2)

给定一个马尔可夫决策过程 MDP ，给定策略 π ，策略 π 的价值（当前状态为 s 时）为 $V_\pi(s) = R(s, a) + \sum_{s'} T(s, a, s')[R(s', \pi(s')) + \sum_{s''} T(s', a, s'')[R(s'', \pi(s'')) + \dots]]$ 。



这两种策略价值的定义是等价的，分别从不同的角度理解策略的价值，并给出了计算方法。定义 8.8 是估计了策略 π 从状态 s 出发的各种长期收益的期望值。

而追求最优策略是很多问题的最终目标，最优策略如定义 8.9 所示。

定义 8.9 (最优策略)

给定一个马尔可夫决策过程 MDP ，给定初态 s ，对任意策略 π ，记其策略价值为 $V_\pi(s)$ ，那么最优策略 $\pi^* = \arg \max_\pi V_\pi(s)$ 。



最优策略就是所有策略中策略价值最大者。

8.4 策略评估

如何求一个马尔可夫决策过程的最优策略？这个任务分两步完成，如下：

- (1) 计算任意一个给定策略的价值；
- (2) 设计基于策略价值比较的最优策略搜索算法。

其中，计算任意一个给定策略 π 的价值是基础，先考虑计算一个策略的价值，最直观的做法是基于定义的方法：知识点8.4。

知识点 8.4 (基于定义的策略价值计算：枚举法)

给定一个马尔可夫决策过程 MDP ，及其一个给定策略 π 和初态 s ，计算策略价值 $V_\pi(s)$ 的过程如下：

- (1) 列出所有随机路径；
- (2) 计算每条随机路径的收益 u_i ；
- (3) 计算每条随机路径的出现概率 p_i ；
- (4) 求期望 $V_\pi(s) = \sum_i p_i u_i$ 。



枚举法存在的问题：随机路径可能有无穷多条，无法枚举所有；因此也无法枚举所有路径的收益和出现概率；例如随机数游戏，可能永远也不会退出，可以生成任意长度的路径；基于策略价值定义的枚举法很难在工程上实现。

例题 8.1（随机数游戏的策略价值计算）设随机数游戏的一个策略是 $\pi = \{\text{游戏中: 继续}, \text{结束: } *\}$ ，其中 * 表示“继续”和“退出”都可以，计算策略 π 的收益 V_π 。

解 若策略 π 的价值收敛，那么应该满足递推关系式： $V_\pi(\text{游戏中}) = 30\% \times 4 + 70\% \times (4 + V_\pi(\text{游戏中}))$ ，解之，得到 $V_\pi(\text{游戏中}) = 40/3$ 。

将求解8.1中的方法一般化，我们得到贝尔曼方程。

知识点 8.5 (Bellman/贝尔曼方程)

给定一个马尔可夫决策过程 MDP ，及其一个给定策略 π 和状态 s ，定义贝尔曼方程为： $V_\pi(s) = \sum_{s' \in \mathbb{S}} T(s, a, s') [U(s, a, s') + V_\pi(s')]$



知识点8.5中的 $V_\pi(s), V_\pi(s')$ 分别表示策略 π 在初态是 s, s' 时的价值。策略 π 是确定性的，故 π 给定时， $a = \pi(s)$ 。

贝尔曼方程也可写成如下的几种形式：

- (1) $V_\pi(s) = R(s, a) + \sum_{s' \in \mathbb{S}} T(s, \pi(s), s') V_\pi(s')$
- (2) $V_\pi(s) = R(s, \pi(s)) + \sum_{s' \in \mathbb{S}} T(s, \pi(s), s') V_\pi(s')$
- (3) $V_\pi(s) = \sum_{s' \in \mathbb{S}} T(s, a, s') U(s, a, s') + \sum_{s' \in \mathbb{S}} T(s, a, s') V_\pi(s')$
- (4) $V_\pi(s) = \sum_{s' \in \mathbb{S}} T(s, \pi(s), s') U(s, \pi(s), s') + \sum_{s' \in \mathbb{S}} T(s, \pi(s), s') V_\pi(s')$

状态空间 \mathbb{S} 中任意状态都可以为初态，建立上述贝尔曼方程，得到大量贝尔曼方程，如知识点8.6所述的贝尔曼方程组。

知识点 8.6 (贝尔曼方程组)

给定一个马尔可夫决策过程 MDP , 及其一个给定策略 π , 设状态空间 $\mathbb{S} = \{s_1, s_2, \dots\}$, 对任意 $s_i \in \mathbb{S}, i = 1, 2, \dots$, 建立方程组如下:

$$V_\pi(s_1) = \sum_{s' \in \mathbb{S}} T(s_1, a, s')[U(s_1, a, s') + V_\pi(s')]$$

$$V_\pi(s_2) = \sum_{s' \in \mathbb{S}} T(s_2, a, s')[U(s_2, a, s') + V_\pi(s')]$$

... ...

这样的方程共有 $|\mathbb{S}|$ 个, 其中的未知数是 $V_\pi(s_i)$, 也正好是 $|\mathbb{S}|$ 个, 构成 $|\mathbb{S}|$ 个 $|\mathbb{S}|$ 元方程组, 该方程组是线性方程组, 一般情形下是有解的。



解贝尔曼方程组, 对任意初态 s , 求得 $V_\pi(s)$ 。线性方程组有多种求解方法, 这里给出一种求解递推方程组的近似算法。

Algorithm 8.1: 求解递推方程组的近似算法

1 Function Policy-Value-Recurs(mdp, π)

Input: 马尔可夫决策过程 mdp ; 策略 π

Output: 策略收益: $V_\pi(s_1), V_\pi(s_2), \dots$

2 $V_\pi \leftarrow 0, i = 1, 2, \dots$ /* 对所有初态, 初始化值函数为 0 */

3 repeat

4 | foreach $s \in \mathbb{S}$ **do**

5 | | $V_\pi(s) \leftarrow \sum_{s' \in \mathbb{S}} T(s, a, s')[U(s, a, s') + V_\pi(s')]$ /* 利用贝尔曼方程更新值函数 */

6 until T 次

算法8.1的停止条件设置为循环 T 次, 也可以设置为 $V_\pi(s)$ 的值相邻两次更新足够小, 例如小于预定义的常数阈值 $\epsilon (> 0)$ 。算法8.1所需的空间大小为 $O(|\mathbb{S}|)$, 即每个状态只需要存储最近一次的 $V_\pi(s)$ 的值。算法8.1花费的时间代价为 $O(|\mathbb{S}|^2 \times T)$ (最坏情形下)。

如果使用算法8.1求解马尔可夫决策过程中一个策略 π 的价值, 那么算法的时空复杂度要求问题的规模不能太大, 即状态空间大小受到限制。

8.5 寻找最优策略

策略评价将一个策略函数 π 转换为一个实数值“策略价值” V_π 。而所有可能策略的集合, 即策略空间, 其大小为 $|\mathcal{A}|^{|\mathbb{S}|}$ 或 $\prod_s |A_s|$, 是状态数目的指数函数, 这种复杂性导致我们不可能让算法评估所有的策略, 然后选择出最优策略。

为讨论寻找最优策略的算法, 我们先引入 Q 值的概念。

定义 8.10 (Q 值)

给定一个马尔可夫决策过程 MDP , 及其一个给定策略 π , Q 值记为 $Q_\pi(s, a)$ 表示初态为 s , 策略 “ $\pi \setminus \{s : \pi(s)\} \cup \{s : a\}$ ” 的策略价值。



定义8.10中策略 “ $\pi \setminus \{s : \pi(s)\} \cup \{s : a\}$ ” 从原始策略 π 中删除了面临状态 s 时的动作决策 $\pi(s)$, 换之以动作 a , 即 $\{s : \pi(s)\} \leftarrow \{s : a\}$ 。故有, 若 $a == \pi(s)$, 那么 $Q_\pi(s, a) == V_\pi(s)$; 若 $a \neq \pi(s)$, 那么 $Q_\pi(s, a)$ 的策略在面临状态 s 时发生改变, 因此与采用原始策略 π 的 $V_\pi(s)$ 值通常不一样。

有了 $Q_\pi(s, a)$ 的定义，可以将 $V_\pi(s)$ 改写成 Q 值的形式，即 $V_\pi(s) = Q_\pi(s, \pi(s))$ 。

$V_\pi(s)$ 是策略 π 和初态 s 的函数，而 $Q_\pi(s, a)$ 是策略 π 、初态 s 和动作 a 的函数。 $V_\pi(s)$ 相当于将 Q 值中策略 π 和某个动作 a^* 用公式 $a^* = \pi(s)$ 固定了下来，而 $Q_\pi(s, a)$ 中的 a 是可以从动作空间 A 中任意取值的自由变量。

引入 Q 值是为了更好地描述最优策略搜索算法。先给出策略改进算法8.2，输入任意一个策略 π 和状态 s ，输出改进的策略 π_{new} 保证面临状态 s 时返回最优动作。

Algorithm 8.2: 策略改进算法

```

1 Function Policy-Promotion( $mdp, \pi$ )
  Input: 马尔可夫决策过程  $mdp$ ; 策略  $\pi$ 
  Output: 改进的策略  $\pi_{new}$ 
2 foreach  $s \in S$  do
  3   foreach  $a \in A$  do
  4     评估策略  $Q_\pi(s, a)$ 
  5    $\pi_{new} \leftarrow \arg \max_{a \in A} Q_\pi(s, a)$ 
```

算法8.2第3, 4, 5行的一次执行会输出面临状态 s 的最优动作。算法8.2最费时间的代码是第4行的评估策略过程，需要解贝尔曼方程组，要执行 $|A| \times |S|$ 次贝尔曼方程组的求解，算法时间复杂度相当高；降低时间代价的一个方法是不评估所有状态的的 $Q_\pi(s, a)$ 值，这个方法称为“异步”策略改进算法。

有了策略改进算法8.2，可以基于它实现对最优策略的搜索。如算法8.3所示的策略迭代算法。

Algorithm 8.3: 策略迭代算法

```

1 Function Policy-Iteration( $mdp$ )
  Input: 马尔可夫决策过程  $mdp$ 
  Output: 最优策略  $\pi^*$ 
2  $\pi^* \leftarrow$  任意值初始化
3 repeat
  4   评估策略  $\pi^*$ , 计算  $V_{\pi^*}$ 
  5    $\pi^* \leftarrow$  Policy-Promotion( $mdp, \pi^*$ )
6 until  $T$  次
```

算法8.3的停止条件设置为循环 T 次，也可以设置为策略 π 不再更新时停止。算法8.3能保证收敛到全局最优解，时间开销与初始解、状态空间大小、动作空间大小，循环次数 T 等因素相关。

算法8.3存在的问题是需要“精确”评估每个“经历过”的策略的价值，浪费了大量的时间，应尽量避免这些计算。因此，由改进的值迭代算法8.4。

Algorithm 8.4: 值迭代算法

```

1 Function Value-Iteration(mdp)
  Input: 马尔可夫决策过程 mdp
  Output: 最优策略  $\pi^*$ 
2  $V_0^* \leftarrow 0$                                 /* 初始化所有状态的最优值函数  $V^*$  为 0 */
3 foreach  $t \in \{1, 2, \dots\}$  do
  4   foreach  $s \in \mathbb{S}$  do
    5      $V_t^*(s) \leftarrow \max_{a \in Action(s)} \sum_{s'} T(s, a, s')[U(s, a, s') + V_{t-1}^*(s')]$ 

```

值迭代算法8.4将策略评估和策略改进两个独立的过程结合在一起，放入一个过程中完成，同样保证了收敛到最优策略，对中间经历过的策略没有进行完整的评估。算法8.4输出最优值函数所对应的策略就是最优策略 π^* 。

❀ 第八章 习题 ❀

1.

第九章 强化学习

内容提要

- 优化问题的例子
- 求解优化问题
- 优化问题搜索形式化
- 优化问题的应用

9.1 优化问题的例子

9.2 优化问题搜索形式化

9.3 求解优化问题

9.4 优化问题的应用

第九章 习题

1.

第十章 博弈与对抗搜索

内容提要

- 最佳优先搜索
- 启发式函数
- 设计启发式函数
- A^* 算法

10.1 最佳优先搜索

10.2 启发式函数

10.3 启发式函数的设计

10.4 A^* 算法

第十章 习题

1.

第四部分

机器学习

第十一章 什么是机器学习

内容提要

- 机器学习的输入
- 机器学习的输出
- 机器学习的处理流程
- 理解机器学习

站在人工智能的角度，描述世界的状态转移模型认为世界在不停地做“状态 \Rightarrow 状态”的转移，如图11.1所示，智能体参与转移过程是通过做出一个“决策/动作”来实现的，这个参与可能是对转移过程的干扰或控制。

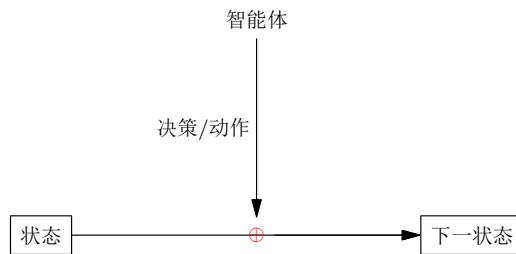


图 11.1: 智能体参与的状态转移模型

假设智能体在给定状态下做的动作对应一个确定的、唯一的下一状态/次态，这样(状态, 动作)和(状态, 次态)就表示了同一个转移。这使得我们对状态转移模型的($\text{状态} \Rightarrow \text{状态}$)的探讨可以通过对($\text{状态} \Rightarrow \text{动作}$)的探讨来实现。即，若探讨世界存在的某个状态转换，不妨设之为 $x' = g(x)$ ，其中 x , x' 表示状态，这是状态转移模型的描述，可以改变成对智能体的描述 $y = f(x)$ ，其中 x 是描述世界的状态， y 是智能体的决策/动作，智能体通过动作 y 参与了状态转移后，会转移到唯一确定的状态 x' 。

这样，对世界各种转换的理解就变成了对智能体 $y = f(x)$ 的设计和制造。作为人工智能核心的智能体 $y = f(x)$ ，其功能就是将一个环境/外部刺激/状态 x 转化为最佳应对的决策/动作 y ，这个决策/动作 y 将使得世界从一个状态转换到另一个状态。

我们把设计和制造智能体 $y = f(x)$ 的过程称之为机器学习。学习是指个体通过感知、思考、实践等在内的各种途径获得知识或技能的过程，学习的直接结果是个体技能或知识的提升，具体表现为个体之后行为方式的可能改变。机器学习中生物个体为机器所替代，而机器现今通常指计算机或人工智能中的智能体。智能体得到更新的 $y = f(x)$ ，决策/行动依据新的 $y = f(x)$ 就会发生改变。

如图11.2所示，给出了机器学习的定义。

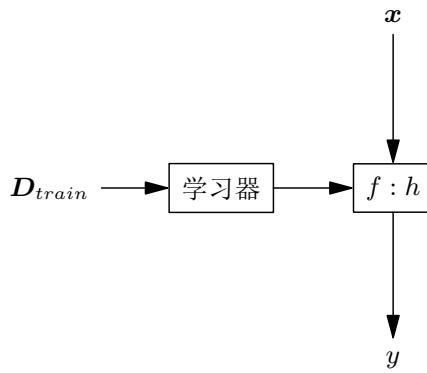


图 11.2: 机器学习的定义

其中 $D_{train} = \{(x, y)\}$ 被称之为训练数据集，是观测到的一些状态转换的例子或事实，被当作学习器的输入，而学习器就是我们研究的机器学习算法，学习器的输出是表示世界真实状态转换过程的完美智能体 $y = f(x)$ 的近似转换 $\tilde{y} = h(x)$ ，表示改进了或更新了的智能体。

从计算机处理信息的角度来看，机器学习的学习器就是计算机信息处理的算法。因此，理解和分析机器学习，我们首先要明确机器学习算法的输入、输出和处理流程。

机器学习的输入可能是过去出现的某些事物，也可能是已经发现的知识和技能；而输出可能就是“改良”或“改进”的新的知识或新技能；处理流程只要能实现从输入到输出的转换，并经过实践的检验，就可能被接受。

11.1 机器学习的输入

- 粗略地说，描述世界的状态，可以是任意的事物做为机器学习的输入，包括人，物，事等；
- 精准地说，任意的事物被各种传感器所感知，生成各种信号和数据，这些信号和数据就是机器学习的输入。

更具体地说，如图11.2中所示的 D_{train} ，称为训练数据集，是机器学习的输入。通常我们把机器学习的输入称为经验或事实，包含我们研究和关注对象的相关信息，正式的名称是数据集和样本。

定义 11.1 (样本)

与调查和研究对象总体相关的部分完整或不完整个体。



例 11.1 (样本) 所有尺寸为 1000×1000 的的图像是我们调查的总体，这种图像的总数有 $256^{1000 \times 1000}$ 张（假设每个像素点有 256 中颜色可选），给定其中 $N (\leq 256^{1000 \times 1000})$ 张图像，获得一个样本/数据集 $D = \{d_1, d_2, \dots, d_N\}$ ，其中 N 是样本容量/样本大小， d_i 表示 D 中的第 i 张图像，也称为一个数据。

常说的大样本和小样本是个相对的概念，与数据的维度相关。比如，10 个人的身高数据集，常被认为是个小样本，1000 个人的身高数据集常被认为是个大样本；1000 个人的人脸图像 (1000×1000) 数据集却可能被视为一个小样本，数十万、上百万的人脸图像数据集才有可能被认为是大样本。

例 11.2 (数据) 一张图像即一个数据， $d \in D$ 是一个数据，并表示为 $d = \{x_{ij}\}_{1000 \times 1000}$ 。该数据包含 10^6 个点，每个点有 256 中可能的色彩取值。数据 d 的维度是 10^6 。

例 11.3 (大数据) 大数据即数据“很大”，即数据的维度很高。通常数据维度超过 100，即可认为是大数据。因此，上例中的每张图像都可视为一个大数据。

海量数据是指数据集中数据个数多，但是并不一定是一个大样本。

可以进一步细分机器学习的输入数据集。假设数据集 $\mathbf{D} = \{\mathbf{d}_1, \mathbf{d}_2, \dots, \mathbf{d}_m\}$ ，第 i 个数据记为(列)向量 $\mathbf{d}_i = (x_{1i}, x_{2i}, \dots, x_{ni})^T$ ，其中 $x_{ji}, j = 1, 2, \dots, n$ ，可以是任何的标量值，比如字符，数字，色彩等等，那么数据集 \mathbf{D} 可以写成公式 (11.1) 所示的矩阵。

$$\mathbf{D} = \begin{pmatrix} \mathbf{d}_1 & \mathbf{d}_2 & \cdots & \mathbf{d}_m \\ x_{11} & x_{12} & \cdots & x_{1m} \\ x_{21} & x_{22} & \cdots & x_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nm} \end{pmatrix} \begin{matrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{matrix} \quad (11.1)$$

公式 (11.1) 中的 $x_i, i = 1, 2, \dots, n$ 称为数据的特征或属性。

定义 11.2 (聚类问题)

若要找如公式 (11.1) 所示数据集 \mathbf{D} 中数据特征之间的隐函数关系 $k = f(x_1, x_2, \dots, x_n), k \in \mathbb{N}^+$ ，我们称之为“聚类问题”。



聚类问题中的 k 可以是来自任何的离散值的集合，例如字母集合 $\{'a', 'b', 'c'\}$ ，或数字集合 $\{1, 2, 3, 4\}$ 等等，关键在于来自这些集合的 k 不是公式 (11.1) 中的任何一个属性。聚类问题也称为无监督学习。

如果从公式 (11.1) 所示数据集 \mathbf{D} 中选择某列出来，重新记作 y ，例如令 $y = x_1$ ，并称 y 为标签或监督信号，而依然称 (x_2, x_3, \dots, x_n) 为特征，那么我们得到有标签的数据集。

为了记号的简单，新的有标签的数据集我们依然记其特征/属性的个数为 n ，得到如公式 (11.2) 所示的数据集 \mathbf{D} 。

$$\mathbf{D} = \begin{pmatrix} \mathbf{d}_1 & \mathbf{d}_2 & \cdots & \mathbf{d}_m \\ y_1 & y_2 & \cdots & y_m \\ x_{11} & x_{12} & \cdots & x_{1m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nm} \end{pmatrix} \begin{matrix} y \\ x_1 \\ \vdots \\ x_n \end{matrix} \quad (11.2)$$

定义 11.3 (分类问题和预测问题)

若要找如公式 (11.2) 所示数据集 \mathbf{D} 中数据特征之间的显式函数关系 $y = f(x_1, x_2, \dots, x_n)$ ，当 y 是离散值时，我们称之为分类问题；当 y 是连续值时，我们称之为预测问题。



定义 11.3 所描述的问题统称为有监督学习，有监督的含义就指数据集中有监督信号 y ；无监督学习则反之，如定义 11.2。现实应用中大多数数据集都没有提供监督信号，或者监督信号不全，或者监督信号太少等；而监督信号的获得会受各种资源条件的限制，如何获得大量、优质的监督信号，成为近年来机器学习领域最热门的研究方向之一。

当我们从计算机编程的角度来看，机器学习的输入通常以自然语言文本、语音、数值、图像和视频等媒介形式提供。

自然语言处理领域里的机器学习，其输入是自然语言，例如中文、英文等的文本或字符串；计算机视觉是指机器学习的输入是图像；当语音作为机器学习的输入时，得到新类型的学习任务；早期机器学习主要研究输入的数值数据；现在机器学习研究更复杂的数据，如关系型数据和多通道数据，典型代表包括图/网络和视频等。

定义 11.4 (多模态学习)

多模态学习是指输入数据包括两种及两种以上媒介形式的机器学习；一种媒介提供数据对象一种对应“模态”的特征。



典型的媒介包括：文本、数值、图像、音频、网络和视频等。现代物联网技术的发展，能提供对关注或研究对象的各类声、光、电、触等感知信号，每类信号能描述数据对象的不同方面，对最终任务的作用或大或小；信号可能在不同类型的媒介上展示出来。

另一个容易混淆的相似概念是多通道学习。

定义 11.5 (多通道学习)

多通道学习是指输入数据来自多个不同的源的机器学习；每个源是一个通道。



比如三个不同位置的相机对同一个物体，同时各自拍下一张照片，这三张照片作为机器学习的输入数据。多通道数据可能来自不同空间、时间和感知设备，故也可以表述为“多源数据”；而多模态更侧重于数据输入机器学习算法时的媒介形式，不强调是来自于一个源还是多个源；多通道的数据可能是同模态的，也可能是不同模态的。

不管是多模态还是多通道，都使得机器学习的输入变得越来越复杂，输入数据的维度、类型呈增大态势，导致“大数据”的出现和流行。大数据的另一个特点是随着特征维度的增加，每个特征对于任务的相对重要性可能都在整体下降；也可能有相当多的特征与任务完全无关，这常被描述成数据的“价值稀疏性”。

11.2 机器学习的输出

机器学习是为了设计和制造完美智能体 $y = f(\mathbf{x})$ 代表真实世界的转换。当我们从发生过的事实和经验，也就是数据集/样本中获得了函数 $\tilde{y} = h(\mathbf{x}) \sim f(\mathbf{x})$ 的解析表达式后，就是设计和制造了一个完美智能体 f 的近似智能体 $\tilde{y} = h(\mathbf{x})$ 。

机器学习的输出 h 有很多名称，比如智能体、假设、模型、真实函数 f 的近似值等等。设计完美智能体 f 的过程，可建模为搜索问题。每个不同的 h 就是一个不同的状态，我们需要找到的目标状态是完美智能体 f ，所有的状态，即不同的 h ，构成了状态空间。这样的状态空间有多大？

假设完美智能体 $y = f(\mathbf{x})$ ，其中 $\mathbf{x} = (x_1, x_2, \dots, x_n) \in \mathbb{D} = \mathbf{D}_1 \times \mathbf{D}_2 \times \dots \times \mathbf{D}_n$ ， $y \in \mathbf{L}$ 。我们仅考虑 $\mathbf{D}_i, i = 1, 2, \dots, n$ 为有限集合的情形，并记变量 x_i 的取值域的大小为 $2 \leq |\mathbf{D}_i| < +\infty$ 。若某个 \mathbf{D}_i 是一个连续区间，我们在计算机工程实践中会首先将其离散化为有限集合。我们可以将完美智能体 $y = f(\mathbf{x})$ 写成如表11.1的形式，其中 $\mathbf{D}_i = \{v_{i1}, v_{i2}, \dots, v_{i|\mathbf{D}_i|}\}, i = 1, 2, \dots, n$ 。

x_1	x_2	...	x_n	$f(x_1, x_2, \dots, x_n)$
v_{11}	v_{21}	...	v_{n1}	y_1
v_{11}	v_{21}	...	v_{n2}	y_2
...
$v_{1 \mathbf{D}_1 }$	$v_{2 \mathbf{D}_2 }$...	$v_{n \mathbf{D}_n }$	$y_{ \mathbf{D}_1 \mathbf{D}_2 \dots \mathbf{D}_n }$

表 11.1: 完美智能体 $y = f(\mathbf{x})$

机器学习的输出 h 是形如表11.1的表, h 和 f 的前 n 列(定义域)一样,但是在最后一列 $f(x_1, x_2, \dots, x_n)$ 上可能有部分不同。若 y 的值域 \mathbf{L} 是有限的离散集合, 即分类问题, 我们记 y 的值域的大小为 $|\mathbf{L}|$; 若 y 的值域 \mathbf{L} 是一段连续区间, 即预测问题, 我们将 \mathbf{L} 离散化为有限集合, 我们也用 $|\mathbf{L}|$ 标记 y 的值域的大小。这样可以得到不同的 h 的总个数为 $|\mathbf{L}|^{|\mathbf{D}_1||\mathbf{D}_2| \dots |\mathbf{D}_n|} \geq 2^{2^n}$, 其中 $2 \leq |\mathbf{L}| \leq +\infty$, 即搜索完美智能体 f 的搜索问题, 其状态空间的大小是 n 的指数函数。

如果机器学习的输出是以形如表11.1的方式呈现, 这样的函数具有最大的描述复杂性, 同时该函数没有利用任何的先验知识。如果我们事先设置一个带参数的函数解析式来压缩表示函数 f 或 h , 那么这个解析式就体现了某种形式的先验知识。例如, 假设 y 和 \mathbf{x} 之间呈线性关系: $y = \mathbf{W} \cdot \mathbf{x} + b$, 其中 \mathbf{W} 和 b 是线性表达式的系数/参数; 这个假设就是先验知识“输入 \mathbf{x} 和输出 y 之间有线性关系存在”。先验知识也可以理解为压缩函数描述长度的方法。

11.3 机器学习的处理流程

早期比较流行的机器学习的处理流程分为: 数据预处理, 模型参数学习和验证评估三步; 现在很难将前两个步骤数据预处理和模型参数学习区分开。

数据预处理包括数据集成、数据清理、数据转换和数据规约, 前两步“数据集成”和“数据清理”和具体的应用问题密切相关, 这里不做讨论, 只关注这两步已经完成后的处理。

数据转换和数据归约在深度学习流行之后, 一个新的技术名称“表示学习”或“表征学习”涵盖了这两部分的预处理工作。更接近表征学习的传统说法是“特征提取”或“特征工程”, 早期并没有冠以“学习”的说法。

当数据预处理完成, 也就是完成了表征学习, 通常会选择一个简单的、带参数的解析函数表达式 h 当作分类或预测任务的模型; 再通过定义一个优化问题, 来确定带参解析函数表达式 h 的系数或参数, 得到解析表达式的完整描述, 这称为模型参数学习。简而言之, 模型参数学习是个两步过程, 首先利用先验知识确定如何压缩 f 的描述长度(选择参数模型), 然后定义优化问题求模型参数。

最后, 需要验证和评估找到的解析表达式 h 是否准确; 或者是找到多个不同的 h , 然后评估它们并选择其中一个作为最终输出。

11.4 h 的准确性

机器学习输出完美智能体 h 的近似 h , 评价 h 的指标就是评价函数的三个指标: 完备性、准确性和复杂性。这一节我们考虑准确性, 准确性和模型参数求解的优化问题定义密切相关。

最准确和最基础的准确性评价方法将表11.1所示的完美智能体 f 和机器学习的输出 h 进行逐一比对, 将二者的误差/不同统计出来, 用于衡量准确性。误差越小, 准确性越高。常见的误差定义包括绝对误差、平方误差和计数误差等。

定义 11.6 (绝对误差/绝对误差均值)

给定完美智能体 f 和机器学习的输出 h , 二者的绝对误差为: $\sum_{\mathbf{x} \in \mathbb{D}} |f(\mathbf{x}) - h(\mathbf{x})|$, 其中 $\mathbb{D} = \mathbf{D}_1 \times \mathbf{D}_2 \times \dots \times \mathbf{D}_n$ 是输入 \mathbf{x} 的值域; 绝对误差均值是对绝对误差求平均: $err_1(h, f) = \frac{1}{|\mathbb{D}|} \sum_{\mathbf{x} \in \mathbb{D}} |f(\mathbf{x}) - h(\mathbf{x})|$ 。

绝对误差适合于 y 取连续值的情形，比如股票价格。

定义 11.7 (平方误差/平方误差均值)

给定完美智能体 f 和机器学习的输出 h ，二者的平方误差为： $\sum_{x \in \mathbb{D}} (f(\mathbf{x}) - h(\mathbf{x}))^2$ ，其中 $\mathbb{D} = \mathbf{D}_1 \times \mathbf{D}_2 \times \dots \times \mathbf{D}_n$ 是输入 \mathbf{x} 的值域；平方误差均值是对平方误差求平均： $err_2(h, f) = \frac{1}{|\mathbb{D}|} \sum_{x \in \mathbb{D}} (f(\mathbf{x}) - h(\mathbf{x}))^2$ 。



平方误差适合于 y 取连续值的情形。

定义 11.8 (计数误差/计数误差均值)

给定完美智能体 f 和机器学习的输出 h ，二者的计数误差为： $\sum_{x \in \mathbb{D}} 1[f(\mathbf{x}) \neq h(\mathbf{x})]$ ，其中 $\mathbb{D} = \mathbf{D}_1 \times \mathbf{D}_2 \times \dots \times \mathbf{D}_n$ 是输入 \mathbf{x} 的值域；计数误差均值是对计数误差求平均： $err_0(h, f) = \frac{1}{|\mathbb{D}|} \sum_{x \in \mathbb{D}} 1[f(\mathbf{x}) \neq h(\mathbf{x})]$ 。



计数误差适合于 y 取离散值的情形。

知识点 11.1 (准确性计算的时间复杂性)

假设一个输入的误差计算代价为 1，那么计算 h 和 f 的误差所花费的时间为 $O(|\mathbf{D}_1 \times \mathbf{D}_2 \times \dots \times \mathbf{D}_n|) \geq O(2^n)$ 。



因此，准确计算两个函数的差异用于评估两个函数的是否一致，工程上存在困难，故有近似计算的方法如知识点 11.2，从已知 f 值的数据集中划分一个子集，称之为“测试数据集” \mathbf{D}_{test} ，检测测试数据集 \mathbf{D}_{test} 上的误差均值，来近似真实的误差均值。

知识点 11.2 (近似计算 h 的准确性)

选择 $\mathbb{D} = \mathbf{D}_1 \times \mathbf{D}_2 \times \dots \times \mathbf{D}_n$ 的一个子集，并命名为 \mathbf{D}_{test} ，且 $\mathbf{D}_{test} \cap \mathbf{D}_{train} = \emptyset$ ，计算误差均值的近似值：

$$\begin{aligned}\overline{err}_0(h, f) &= \frac{1}{|\mathbf{D}_{test}|} \sum_{x \in \mathbf{D}_{test}} 1[f(\mathbf{x}) \neq h(\mathbf{x})] \sim err_0, \\ \overline{err}_1(h, f) &= \frac{1}{|\mathbf{D}_{test}|} \sum_{x \in \mathbf{D}_{test}} |f(\mathbf{x}) - h(\mathbf{x})| \sim err_1, \\ \overline{err}_2(h, f) &= \frac{1}{|\mathbf{D}_{test}|} \sum_{x \in \mathbf{D}_{test}} (f(\mathbf{x}) - h(\mathbf{x}))^2 \sim err_2.\end{aligned}$$



利用知识点 11.2 评估 h 的准确性时，还有隐含一个条件：数据在 \mathbf{D}_{test} 中的分布要保持和在 \mathbb{D} 中的分布一致，但是这个条件很难验证。所以，通常降低要求，保证 \mathbf{D}_{test} 和 \mathbf{D}_{train} 的分布一致，且二者的交集为空即可。而更常见的做法是随机从一个数据集中均匀随机划分出 \mathbf{D}_{test} 和 \mathbf{D}_{train} ，并不做进一步验证，这有一定的概率导致准确性的错误评估。

11.5 理解机器学习

知识点 11.3 (机器学习就是完善表格)

将公式 11.2 所示数据矩阵进行转置，调整成形如表格 11.1 所示的表格，那么机器学习就是从这个 $m (\ll |\mathbf{D}_1 \times \mathbf{D}_2 \times \dots \times \mathbf{D}_n|)$ 行的表格中寻找规律，将其扩充、完善成完美智能体 $y = f(\mathbf{x})$ （表格 11.1）的过程。



如果机器学习得到的表格11.1太大，机器/智能体存储空间无法容纳该表格，那么将其描述成一个数学解析式，也是机器学习的目标。

知识点 11.4 (机器学习就是将表格描述为函数解析式)

将公式11.2所示数据矩阵表示成给定形式的函数解析式。



函数的解析式形式千变万化，数之不尽；然而表格11.1所示的完美智能体，可能的近似函数 h 却是有限可数的 $|L|^{|\mathcal{D}_1||\mathcal{D}_2| \dots |\mathcal{D}_n|}$ 个。当这些无穷尽的函数解析式离散化为 h 时，会有很多看上去不一样，本质上（计算机内部的表格描述）却一样的函数。机器学习就是要找一个和完美智能体（表格11.1）一致且最简单的函数解析式。给定形式的函数解析式，比如线性形式，多项式函数或三角函数等，需要利用数据矩阵中的数据确定解析表达式的系数或参数。同时，要确保这种函数形式能表达出一切可能的、形如表格11.1的完美智能体 $y = f(\mathbf{x})$ 。很显然，单个线性解析式和单个三角函数并不能表示出各种复杂的函数；神经网络是一类这样的解析表达式，当其节点数、层数等设计合理时，能逼近任意想要的函数 $y = f(\mathbf{x})$ ，即可以离散化为任意想要的形如表格11.1的完美智能体。

知识点 11.5 (机器学习就是解方程组)

将公式11.2所示数据矩阵中的每个数据代入给定形式的函数解析式，得到 m 个方程，求解这 m 个方程构成的方程组。



求解方程组就是计算给定形式的函数解析式的系数。这个方程组可能是超定的、无解的；也可能是欠定的有无穷多组解；也可能正好只有一个解。

方程组正好有一个解时，有可能因为数据噪声的存在，导致解出来的系数不正确。

方程组欠定、有无穷组解时，可能因为给定形式的函数解析式过于复杂，样本太小，函数解析式中有太多的自由参数不能确定，所以此时选择一组系数或一个函数解析式，具有一定的随机性，发生常说的“过拟合”现象。

方程组超定、无解时，有两种解释：

- (1) 给定形式的函数解析式过于简单，表达能力不够，不能够表示出复杂的真实函数 f ；
- (2) 数据采集中引入了噪声，不能满足方程组。

总而言之，只有训练数据集时函数解析式的确定很困难，会造成上述三种情形之一出现；利用经验，设计合适的神经网络结构（不同的神经网络结构代表一种不同的函数解析式）是目前主流的方法。

知识点 11.6 (机器学习就是约束满足问题求解)

当求解的 m 个方程被视为等式约束条件时，机器学习就是约束满足问题求解。



当把机器学习理解为解方程组或者求解约束满足问题时，我们假设给定了带参数的函数解析式。如果把“给定”一词去掉，认为机器学习包含寻找“好的”带参数的函数解析式及其参数两个部分的内容，那么机器学习的内容和变化就变得尤为丰富。

 第十一章 习题

1. 形如11.1的函数 f , 采用计数误差评估其近似值 h 的准确性。若训练样本大小为 m , 若训练误差为 e_1 , h 距离 f 的误差大小为 e_2 , 这样的 h 有多少个?
2. 形如11.1的函数 f , 采用计数误差评估其近似值 h 的准确性。若训练样本大小为 m , 测试集大小为 k , 测试集上的误差均值为 e_0 , h 距离 f 的误差范围是什么? 若用误差均值 e_0 评估函数 h 的准确性, 有什么要求条件?

第十二章 线性回归

内容提要

- 手写体数字识别
- 损失函数
- 特征提取函数
- 最小化损失

线性回归是给定函数 $y = f(\mathbf{x})$ 的解析形式为线性函数，同时 y 的取值为连续值时的问题，是机器学习中最简单的情形。我们从这里开始。

12.1 手写体数字识别

例 12.1 (手写体数字识别) 给定一组手写体数字图片 \mathcal{D}_{train} ，寻找函数 $y = f(\mathbf{x})$ 能将输入手写体数字图片 \mathbf{x} 转化成对应的数字 y 。

如图12.1给出的手写体数字，每个手写体数字图片 \mathbf{x} 是 100×100 的像素点方阵，每个像素的的取值为 0 或 1，1 表示黑点，0 表示白点；即函数 $y = f(\mathbf{x})$ 的输入 \mathbf{x} 是一个 100×100 的 0-1 矩阵；输出 $y \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ 。尽管 y 取值于离散值集合，为简化问题，我们可以令 $y \in \mathbb{R}$ ，这样，手写体数字识别问题就被定义成了预测问题。

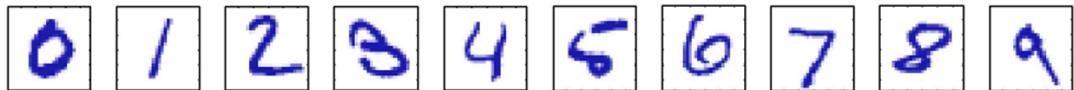


图 12.1: 手写体数字的例子

注 当我们识别数字时，可以依据 y 离数字 $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$ 的远近来离散化，例如 $y = -3.4$ ，就输出 0； $y = 4.2$ ，就输出 4； $y = 200$ ，就输出 9。

问题分析：

- (1) 现实世界的真实转换 f 是什么？人眼视觉感知，看到图片，信号传入大脑皮层，在经历了一系列复杂的视觉、光学、神经信息处理等物理过程后，得到关于输出数字的判决；该 f 目前无法确立一个精确的数学函数描述。
- (2) 机器学习的输出 $h \sim f$ 是什么？这里，给定最简单的函数解析式： $h(\mathbf{x}) = \mathbf{W} \cdot \mathbf{x} + b$ ，这是线性表达式，其中 \mathbf{W}, b 是线性表达式的系数，运算“.”表示两个向量或两个矩阵的内积，即两个维度一样的向量或矩阵对应位置元素乘积之和。 h 中 \mathbf{W} 和输入 \mathbf{x} 维度一致，其每个元素的值是对输入图片中对应位置的像素点进行“加权”，描述对应像素点在识别过程中的作用。输入 \mathbf{x} 中每个像素点在每个不同数字的识别中，作用可能各不相同，如何设置该权值成为关键。

搜索最佳 h ：

当 h 的解析式给定为线性表达式 $h(\mathbf{x}) = \mathbf{W} \cdot \mathbf{x} + b$ ，那么一组不同的 \mathbf{W}, b 系数值/参数值代表一个不同的函数 h ，在识别手写体数字时，准确率或高或低，我们要找到准确率最高的那组参数 \mathbf{W}^*, b^* ，并将其确定的 $h(\mathbf{x}) = \mathbf{W}^* \cdot \mathbf{x} + b^*$ 称为最佳 h 。

接下来我们从损失函数的角度来定义优化问题，并搜索最佳参数组，求解 h 。

12.2 特征提取函数

知识点 12.1 (去掉线性解析式的常数项)

线性形式的函数解析式一般都包含常数项，在机器学习中，可以通过简单变换将 h 定义为齐次的线性形式，即：将输入 \mathbf{x} 增加一维，即 $\mathbf{x} = (x_1, x_2, \dots, x_n)^T \rightarrow \mathbf{x} = (x_1, x_2, \dots, x_n, 1)^T$ ，对应线性解析式的参数 $\mathbf{W} = (w_1, w_2, \dots, w_n)^T \rightarrow \mathbf{W} = (w_1, w_2, \dots, w_n, b)^T$ 。

定义 12.1 (特征提取函数)

对函数 $y = f(\mathbf{x})$ 的输入 \mathbf{x} 进行变换，得到新的特征或输入，这样的变换称为特征提取函数，通常记为 $\phi(\mathbf{x})$ ，即 $\mathbf{x} = (x_1, x_2, \dots, x_n)^T \rightarrow \phi(\mathbf{x}) = (\phi_1(\mathbf{x}), \phi_2(\mathbf{x}), \dots, \phi_d(\mathbf{x}))^T$ 。

知识点12.1中的特征提取函数就是 $\phi(\mathbf{x}) = (\phi_1(\mathbf{x}), \phi_2(\mathbf{x}), \dots, \phi_d(\mathbf{x}))^T = (x_1, x_2, \dots, x_n, 1)^T$ ，其中 $d = n + 1$ 。这个的特征提取函数非常简单，仅完成了函数解析式的齐次化功能，但是特征提取函数还可以完成更多更复杂的功能，成为近年机器学习研究最热门的方向之一。

我们称齐次的线性解析式为机器学习中 h 的标准线性表达式，记为 $h(\mathbf{x}) = \mathbf{W} \cdot \phi(\mathbf{x})$ 。

12.3 损失函数

要搜索最佳 h ，即准确率最高的 h 或误差最小的 h 。我们从搜索的目标出发，寻找误差最小的 h 。一个数据上的误差，也称为数据损失。

定义 12.2 (数据损失)

给定函数 f 及其近似值 h ，它们在数据 \mathbf{x} 上的误差或损失为 $f(\mathbf{x})$ 和 $h(\mathbf{x})$ 之间的差异。

如知识点11.2给出了三种误差计算公式，都可以用来定义数据损失，如下：

- (1) 绝对损失: $loss_1(h, f, x) = |f(x) - h(x)|$, 适用于连续 y 值情形
- (2) 平方损失: $loss_2(h, f, x) = (f(x) - h(x))^2$, 适用于连续 y 值情形
- (3) 计数损失: $loss_0(h, f, x) = 1[f(x) \neq h(x)]$, 适用于离散 y 值情形

当 h 的解析式是线性的，即 $h(\mathbf{x}) = \mathbf{W} \cdot \phi(\mathbf{x})$ 时，一个数据的损失对应为：

- (1) 绝对损失: $loss_1(\mathbf{W}, f, x) = |f(x) - h(x)| = |\mathbf{W} \cdot \phi(x) - f(x)|$
- (2) 平方损失: $loss_2(\mathbf{W}, f, x) = (f(x) - h(x))^2 = (\mathbf{W} \cdot \phi(x) - f(x))^2$
- (3) 计数损失: $loss_0(\mathbf{W}, f, x) = 1[f(x) \neq \mathbf{W} \cdot \phi(x)]$

注 本章仅考虑 h 的解析式为线性的情形。

$h(\mathbf{x}) = \mathbf{W} \cdot \phi(\mathbf{x})$ 中， $\mathbf{W} \cdot \phi(\mathbf{x})$ 也有时被称为预测值，基于预测值，我们可以定义残差/residual。

定义 12.3 (残差/residual)

预测值减去真实值即为残差，记为 $residual = \mathbf{W} \cdot \phi(\mathbf{x}) - y$ 或 $residual = \mathbf{W} \cdot \phi(\mathbf{x}) - f(\mathbf{x})$

残差有可能为正，而言可能为负。如图12.2所示，图中红色直线为 $h(\mathbf{x}) = \mathbf{W} \cdot \phi(\mathbf{x})$ ，绿色点 $(\phi(\mathbf{x}), y)$ 为一个真实数据，图中从绿色点到红色直线的虚线长度即为残差大小，直线之上的绿色点对应的残差是负数。

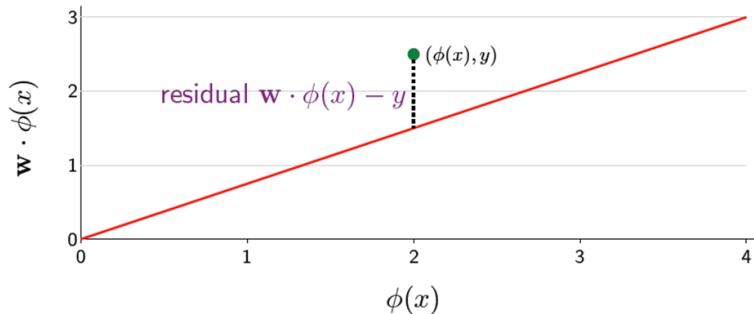


图 12.2: 残差的定义

基于残差可以定义数据损失，得到损失函数。

定义 12.4 (平方损失函数)

$$\text{loss}_2(\mathbf{W}, \mathbf{x}, y) = (h(\mathbf{x}) - f(\mathbf{x}))^2 = (\mathbf{W} \cdot \phi(\mathbf{x}) - y)^2, \text{ 即残差的平方。}$$



定义 12.5 (绝对值损失函数)

$$\text{loss}_1(\mathbf{W}, \mathbf{x}, y) = |h(\mathbf{x}) - f(\mathbf{x})| = |\mathbf{W} \cdot \phi(\mathbf{x}) - y|, \text{ 即残差的绝对值。}$$



定义12.4和定义12.5给出的平方损失函数和绝对值损失函数的图像如图12.3所示，横坐标为残差，纵坐标为损失值/误差。两个损失函数在残差为0时，损失值为0；残差非0时，损失都大于0，且关于y轴 $x = 0$ 对称。

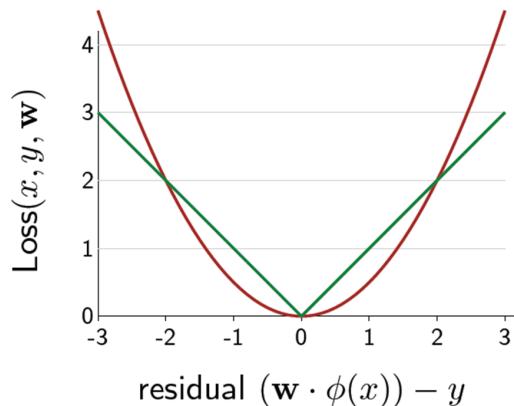


图 12.3: 基于残差定义的平方损失函数（深红色曲线）和绝对值损失函数（绿色曲线）

12.4 最小化损失

从知识点11.5所述的机器学习是求解方程组的观点来看，求解由数据集构造的方程组是一种最直观的求解函数 h 解析式的最优参数方法，但是面临超定或欠定的问题。超定或欠定受函数解析式的复杂性和数据个数两个因素的综合影响，而这两个因素无法随意调整，因此仅需要不同于解方程组的方法求函数解析式的参数。

理论上，最好的 h 在所有数据上和真实函数 f 都保持一致，即 $\sum_{\mathbf{x} \in \mathbb{D}} \text{loss}(\mathbf{W}, y, \mathbf{x}) = \sum_{\mathbf{x} \in \mathbb{D}} (\mathbf{W} \cdot \phi(\mathbf{x}) - y)^2 = 0$ ，我们可以据此定义一个追求误差最小的优化问题，但是完成该误差/损失函数的计算

并求和，在工程上无法实现，时间复杂度太高。所以，我们在训练数据集上计算所有损失之和，并用作优化问题的目标函数，在工程实现上使之可行。

定义 12.6 (定义优化问题：寻找解析式的最佳参数)

给定训练数据集 D_{train} 以及损失函数 $loss(x, y, \theta)$ ，其中 θ 是函数 h 解析式的参数，为求解该参数的最优值，定义优化问题： $\arg \min_{\theta} \sum_{(x,y) \in D_{train}} \frac{loss(x,y,\theta)}{|D_{train}|}$ 。



例 12.2 线性回归问题的优化函数 若函数 h 的解析式是线性的，且损失函数为平方损失函数，那么寻找最佳参数的优化问题为： $\arg \min_{\theta} \sum_{(x,y) \in D_{train}} \frac{loss(x,y,\theta)}{|D_{train}|} = \arg \min_{\theta} \sum_{(x,y) \in D_{train}} \frac{(\theta \cdot \phi(x) - y)^2}{|D_{train}|}$ 。

例12.2中的优化问题是一个关于 θ 的二次函数。该二次函数的最优目标值为 0，在小样本情形下，若参数 θ 的维度较高，比如大于样本中数据个数，那么可能有多组参数值 θ 达到最优目标值 0。但也可能样本中数据个数多于参数个数，且知识点11.5所示方程组无解，此时我们通过寻找一个非零的最小目标函数值来找到最优参数 θ 。

知识点 12.2 (如何选择损失函数)

1. 平方损失函数：近似寻找 y 的均值，综合所有数据的影响；
2. 绝对值损失函数：近似寻找 y 的中值，能更好地抵抗离群点的影响。



不同的损失函数定义就意味着一个不同的线性回归算法。

例 12.3 (线性回归最优参数的求解) 利用梯度下降法求解优化问题： $\arg \min_{\theta} \sum_{(x,y) \in D_{train}} \frac{(\theta \cdot \phi(x) - y)^2}{|D_{train}|}$ 。

$$(1) \text{ 梯度信息为: } \nabla_{\theta} \left(\frac{\sum_{(x,y) \in D_{train}} (\theta \cdot \phi(x) - y)^2}{|D_{train}|} \right) = \frac{\sum_{(x,y) \in D_{train}} 2(\theta \cdot \phi(x) - y) \phi(x)}{|D_{train}|};$$

$$(2) \text{ 参数更新公式为: } \theta_{t+1} = \theta_t - \frac{2\lambda_t}{|D_{train}|} \sum_{(x,y) \in D_{train}} (\theta \cdot \phi(x) - y) \phi(x)$$

例12.2中梯度下降法求最优参数的计算量都聚集在求梯度信息，即对训练数据集求和的部分，即 $\sum_{(x,y) \in D_{train}} (\theta \cdot \phi(x) - y) \phi(x)$ ，这是通过一个循环遍历训练数据集来实现。当训练数据集非常大，无法一次载入内存时，所有训练需数据需要从外存载入内存多次，具体次数是梯度下降算法的迭代次数；这造成了巨大的时间开销。研究梯度下降的时间开销是整个机器学习领域的一个基础性问题。

第十二章 习题

1.

第十三章 线性分类

内容提要

- 电子邮件地址识别
- 线性支持向量机
- 线性分类的几何意义

13.1 电子邮件地址识别

例 13.1 (电子邮件地址识别) 给定一个字符串 x 判定其是否是电子邮件地址, 若是则返回 $y = 1$, 否则返回 $y = -1$ 。

例如 $x = "zhangsan@ustc.edu.cn"$, 对应的 $y = 1$, 表示 x 是一个符合通常意义的电子邮件地址; 若 $x = "abc.def.ghi"$, 则对应的 $y = -1$, 表示 x 不是一个合法的电子邮件地址。例13.1需要寻找一个函数 h 来模拟一个正常智力/知识水平人类识别电子邮件能力。假设人类识别电子邮件地址的能力用函数 f 来描述, 这个识别能力输出是离散的 y 值, 属于对输入 x 的分类问题。

问题分析:

首先对输入 x 进行预处理, 即设计特征提取函数, 完成特征提取。

- (1) x 表示一个字符串, 变化范围大, 非结构化数据, 处理复杂;
- (2) 不妨设特征提取函数形如: $x \Rightarrow \phi(x) = (\phi_1(x), \phi_2(x), \dots, \phi_d(x))$, 输出维度为 d 的向量;
- (3) 特征提取函数的设计和实现现在早期图像和语音等的识别研究中非常重要, 占据相当重要的位置;
- (4) 如图13.1所示的特征提取函数将任意一个字符串转换输出一个维度 $d = 5$ 的实数向量, 如 $(1, 0.85, 1, 1, 1)$, 可视为 5 维空间中的一个点。



图 13.1: 电子邮件地址识别的特征提取

13.2 线性分类的几何意义

13.3 线性支持向量机

第十三章 习题

1.

第十四章 非线性函数的学习

内容提要

- 非线性支持向量机
- k -近邻算法
- 决策树

14.1 非线性支持向量机

14.2 k -近邻算法

14.3 决策树

第十四章 习题

1.

第十五章 贝叶斯网络

内容提要

- 概率论与随机向量
- 贝叶斯网络的概念
- 获得贝叶斯网络
- 不确定性推理
- 应用的例子

15.1 概率论与随机向量

15.2 贝叶斯网络的概念

15.3 获得贝叶斯网络

15.4 不确定性推理

15.5 应用的例子

第十五章 习题

1.

第五部分

重要专题

第十六章 梯度下降与神经网络

第十七章 马尔科夫决策过程与强化学习

第十八章 深度学习

第十九章 生成对抗网络

第二十章 表征学习

第二十一章 自然语言处理

第二十二章 网络数据处理

第二十三章 逻辑、知识表示和推理

第二十四章 多智能体系统

第二十五章 大数据与数据挖掘

第二十六章 注意力机制

第二十七章 胶囊网络

致谢

1. 本书基于 ElegantL^AT_EX 项目组提供的书籍模板完成撰写。ElegantL^AT_EX 项目组致力于打造一系列美观、优雅、简便的模板方便用户使用。目前由 **ElegantNote**, **ElegantBook**, **ElegantPaper** 组成，分别用于排版笔记，书籍和工作论文。
2. 本书内容用于教学，部分引用图表来自互联网，不一一指出原始出处，如有侵犯了原作者知识产权的情形，请告知并联系 jlli@ustc.edu.cn.

第二十八章 基本数学工具

本附录包括了人工智能中用到的一些基本数学，我们扼要论述了线性代数的基本运算和性质。我们还介绍了概率论中的一些基本概念。

28.1 线性代数

实线性空间：定义在数域 \mathcal{F} 上的向量空间是一个集合 \mathbf{V} 及其上定义的如下向量加法和标量乘法运算：

向量加法： $\mathbf{V} + \mathbf{V} \rightarrow \mathbf{V}$, 记作 $\mathbf{v} + \mathbf{w}, \forall \mathbf{v}, \mathbf{w} \in \mathbf{V}$

标量乘法： $\mathcal{F} \times \mathbf{V} \rightarrow \mathbf{V}$, 记作 $a\mathbf{v}, \forall a \in \mathcal{F}, \mathbf{v} \in \mathbf{V}$

并满足以下公理 ($\forall a, b \in \mathcal{F}, \mathbf{u}, \mathbf{v}, \mathbf{w} \in \mathbf{V}$) :

1. 向量加法结合律： $\mathbf{u} + (\mathbf{v} + \mathbf{w}) = (\mathbf{u} + \mathbf{v}) + \mathbf{w}$;
2. 向量加法交换律： $\mathbf{v} + \mathbf{w} = \mathbf{w} + \mathbf{v}$;
3. 向量加法的单位元： \mathbf{V} 里有一个零向量 $\mathbf{0}, \forall \mathbf{v} \in \mathbf{V}, \mathbf{v} + \mathbf{0} = \mathbf{v}$;
4. 向量加法的逆元素： $\forall \mathbf{v} \in \mathbf{V}, \exists \mathbf{w} \in \mathbf{V}$, 使得 $\mathbf{v} + \mathbf{w} = \mathbf{0}$;
5. 标量乘法分配于向量加法上： $a(\mathbf{v} + \mathbf{w}) = a\mathbf{v} + a\mathbf{w}$;
6. 标量乘法分配于域加法上： $(a + b)\mathbf{v} = a\mathbf{v} + b\mathbf{v}$;
7. 标量乘法一致于标量的域乘法： $a(b\mathbf{v}) = (ab)\mathbf{v}$;
8. 标量乘法有单位元： $1\mathbf{v} = \mathbf{v}$, 这里 1 是指域 \mathcal{F} 的乘法单位元;
9. \mathbf{V} 在向量加法下闭合： $\mathbf{v} + \mathbf{w} \in \mathbf{V}$;
10. \mathbf{V} 在标量乘法下闭合： $a\mathbf{v} \in \mathbf{V}$ 。

当数域 \mathcal{F} 为实数域时， \mathbf{V} 被称为实线性空间。

线性空间的扩展：

1. 一个实数或复数向量空间加上长度（范数）概念，就是范数称为赋范向量空间。
2. 一个实数或复数向量空间加上长度和角度的概念，称为内积空间。

内积空间中 n 维向量的几何意义：一个 n 维内积空间点可视为从原点到该点的一个“箭头”，有长度和角度。

线性无关：设 \mathbf{V} 是一个线性空间，如果存在不全为零的系数 $c_1, c_2, \dots, c_n \in \mathcal{F}$, 使得 $c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + \dots + c_n\mathbf{v}_n = 0$, 那么其中 n 个向量 $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n$ 称为线性相关的。

反之，称这组向量 $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n$ 为线性无关的。

线性空间的基：设 \mathbf{V} 是一个线性空间，若一个向量集合 \mathbf{B} 是线性无关的，且 \mathbf{B} 能够生成 \mathbf{V} (包含 \mathbf{B} 的最小子空间是 \mathbf{V} ，即 \mathbf{V} 中任意向量可以表示为 \mathbf{B} 的线性加权和)，就称 \mathbf{B} 为 \mathbf{V} 的一个基。对非零向量空间 \mathbf{V} ，基是 \mathbf{V} 最小的生成集，也是极大线性无关组。有限维向量空间的所有基拥有的基向量个数相同，该个数称为该有限维向量空间的维度。

线性映射/线性变换/线性算子：线性空间 \mathbf{V} 到自身的映射/变换。等价定义（是在两个向量空间之间的保运算函数）：

线性空间 \mathbf{V} 上的一个变换 A 称为线性变换，对于 $\forall \alpha, \beta \in \mathbf{V}$ 和数域 \mathcal{P} 中任意 k ，都有

$$A(\alpha + \beta) = A(\alpha) + A(\beta), \quad A(k\alpha) = kA(\alpha) \quad (28.1)$$

理解 1：两个一样的线性空间，分别记为 \mathbf{V}, \mathbf{W} ，在 \mathbf{V} 中的一个向量 \mathbf{v} 对应到在 \mathbf{W} 中的一个向量 \mathbf{w} ，这种对应/变换关系，记作线性变换 $\mathbf{v} = A\mathbf{w}$ 。线性变换 A 描述了两个采用标准正交基 \mathbf{I} 的线性空间中的两个点 \mathbf{v}, \mathbf{w} 之间的坐标变换方式。

理解 2：向量 $\mathbf{v} \in \mathbf{V}, \mathbf{w} \in \mathbf{W}$ 满足关系 $\mathbf{I}\mathbf{v} = \mathbf{v} = A\mathbf{w}$ ，用两种方法描述了一个向量空间中的一个点 p ，用“基·点在基中的坐标”来描述点，即用标准正交基 \mathbf{I} 中的坐标 \mathbf{v} 描述点 p ，用基 \mathbf{A} 中的坐标 \mathbf{w} 描述点 p 。 $A\mathbf{w}$ 表示点 p 在基 A 中的坐标为 \mathbf{w} 。这种线性变换理解方式是“点不动，变换基”，线性变换 A 是一组基。

矩阵的几何意义：

第二十九章 部分实现代码

```
\documentclass[lang=cn,11pt]{elegantbook}

% title info
\title{Title}
\subtitle{Subtitle is here}

% bio info
\author{Your Name}
\institute{XXX University}
\date{\today}

% extra info
\version{1.00}
\extrainfo{Victory won\rq{}t come to us unless we go to it. --- M. Moore}
\logo{logo.png}
\cover{cover.jpg}

\begin{document}

\maketitle
\tableofcontents
\mainmatter
\hypersetup{pageanchor=true}
% add preface chapter here if needed
\chapter{Example Chapter Title}
The content of \texttt{chapter} one.

\bibliography{reference}

\end{document}
```