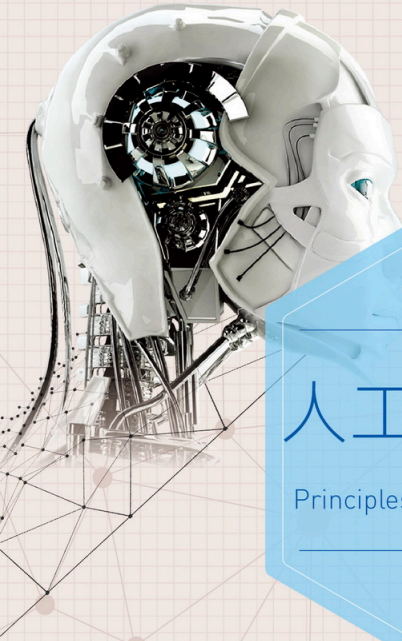
“十三五”国家重点图书出版规划

中国人工智能学会推荐



人工智能丛书

人工智能原理

Principles of Artificial Intelligence

王文敏

高等教育出版社

人工智能丛书编委会

前　言

人工智能被认为是一项对人类社会产生颠覆式影响的科学与技术，其发展之迅猛以及影响面之大，远远超过人们的预料。

人工智能的时代已经到来，准备好去拥抱这个时代了吗？

人工智能于1956年诞生，经历过两次低潮之后，计算能力的迅速提升为其提供了强大的引擎，多媒体数据的爆发性增长为其提供了充足的原料，互联网的迅猛发展为其提供了无垠的空间。人工智能先后战胜了人类的国际象棋、围棋以及德州扑克的顶级选手，图像的识别与分类能力已经超过人类，语音、指纹等识别方式正在改变人机交互的手段，无人驾驶汽车已经在真实环境下试运行数百万公里，双足机器人已经实现了华丽的后空翻。人工智能的学术研究越来越深入，人工智能的创业者越来越多。

套用著名作家狄更斯的一句名言：这是人工智能的“最好时代”，这是人工智能“最有争议的时代”。

世界主要发达国家都把人工智能作为目前最大的发展战略，力图在新一轮国际竞争中掌握主导权。中国国务院于2017年7月8日印发了《新一代人工智能发展规划》，明确提出了中国新一代人工智能实行“三步走”的发展战略，即到2020年人工智能总体技术和应用与世界先进水平同步，到2025年人工智能基础理论实现重大突破，在2030年成为世界主要的人工智能创新中心。这里可以用六个字来概括“三步走”的战略：即同步、突破、领先。所以说，这是人工智能的“最好时代”。

可是，有人担心人工智能会造成大批人失业，有人认为人工智能是人类的威胁，有人游说人工智能可能引发第三次世界大战，更有人惧怕人工智能最终会毁灭人类。所以说，这是人工智能的“最有争议的时代”。

最好也罢，最有争议也罢，人工智能究竟是什么？可以说，人工智能是学术研究，人工智能是工程技术，人工智能是人类科学发展的智慧结晶。

最好也罢，最有争议也罢，人工智能究竟研究什么？可以认为，人工智能研究如何用硬件或软件实现理性的智能，包括智能感知、认知与行为。人类研究人工智能，人类掌握人工智能，人工智能必将造福于人类。

当你在学习人工智能的过程中，你对人工智能的理解会更全面；当你在探索

人工智能的过程中，你对人工智能的认识会更清晰；当你在实践人工智能的过程中，你对人工智能的感受会更深刻。

本书做了一个尝试，即对人工智能的学科及其研究领域进行抽象，梳理出一个人工智能的研究体系，然后按照这个体系讲述其原理，再穿插一些实例分析。

全书分为五篇，共十四章。这五篇分别是：人工智能的“体系篇”“求解篇”“规划篇”“学习篇”以及“推理篇”。

首先，体系篇中有两章。第 1 章是绪论，介绍人工智能的基础、发展史、突破性进展、图灵测试和塞尔的思想实验以及人工智能的三个层级。第 2 章是体系论，在探讨人工智能的本源、内涵与外延的基础上，去粗取精，提纲挈领，给出一个人工智能的研究体系，再将人工智能的诸多研究分支囊括在这个研究体系之中。人工智能首先要研究如何用硬件或软件去实现理性思维，而思维的主要表现形式可以归结为求解、规划、学习以及推理，这是后四篇名称的由来。

其次，求解篇分成四章。本书在人工智能的问题求解中挑选出四类代表性问题，即搜索问题、优化问题、博弈问题以及约束问题，分别在第 3 章至第 6 章论述这四类问题的求解方法。

再次，规划篇中有两章。分别是第 7 章的时空关联规划以及第 8 章的决策理论规划。前者讲述与时间和空间有关的规划方法，而后者是建立在马尔科夫决策过程和动态规划之上的决策理论规划方法。

接着，是学习篇，这里的学习指的是机器学习。本书又将其进行梳理，分成了三个视角，并在第 9 章中对这三个视角进行了论述，然后再用三章的篇幅分别讲述这三个视角，即：机器学习的任务、范式以及框架，机器学习的一些典型算法将归纳到这三个视角之中。这是本书的另一个尝试，即从三个角度和层面来诠释机器学习，而不是仅仅讲述若干个机器学习的代表性算法。

最后，是推理篇。这一篇包含第 13 章知识表示和第 14 章推理机制。在知识表示中，除了介绍一些代表性的知识表示之外，还讲述了经典逻辑、模态逻辑、描述逻辑，以及本体、知识图谱等。在推理中，不仅论述了逻辑推理、定性推理、语义推理以及概率推理，还涉猎了近几年提出的机器推理等。

本书中每个篇章的组织与结构如图 1 所示。

综上，本书提出了一个人工智能的研究体系、三个机器学习的研读视角，并围绕一个体系、三个视角来撰写。这是本书的一个尝试，几点愚见，不当之处，敬请批评指正，共同商榷。借用陶渊明诗中的一句话，“疑义相与析”。

本书可作为人工智能、计算机及其相关专业的高年级本科生或研究生的教材，也可作为人工智能的参考书，供有关科研人员使用。

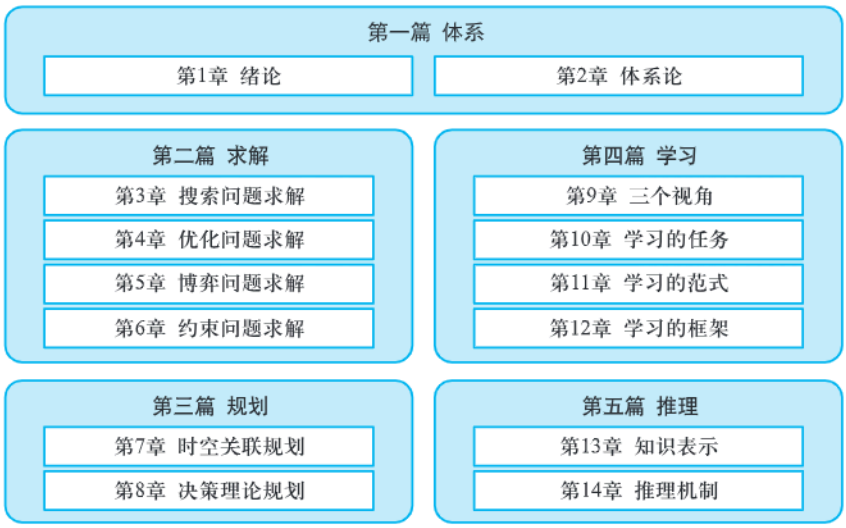


图 1　本书篇章的组织与结构

阅读本书后您会发现，书中用到的英文较多。毋庸讳言，这是为了尊重引用到的原文，为了避免翻译上的差异，也为了读者们查阅英文资料时的方便。

不仅是专业术语，对于一些非第一手信息和数据，我都尽量找到原文加以核对，避免道听途说。本书中涉猎到的外国人姓名，我查阅有关资料选择常用的中文翻译，再将其英文放在后面便于对照。对于原文中出现的外国人名的缩写，我也尽量找到其全名。

尽管学习和研究人工智能多年，又参考了大量的书籍、论文以及英文版的维基百科等，但书中错误难免，欢迎专家和读者给予批评和指正。来函请发至 wang.wenmin@ qq.com。如果本书能成为读者们学习和研究人工智能过程中的一块敲门砖、抑或是一块铺路石，就颇感欣慰了。

王文敏

2019 年 3 月

致　谢

我于1986年在哈尔滨工业大学计算机系攻读博士学位期间，正好赶上了参加国家863计划智能计算机主题中的项目，使我有机会涉足人工智能这个研究领域。我于1988年12月写就的博士论文，题目就是《支持AI问题求解的成员系统模型研究》。1989年又以“支持协同式问题求解的成员系统语言及其并行模型研究”为题获得了国家青年自然科学基金资助，是当时计算机领域获得资助的仅有的几个青年学者之一。弹指间三十余载，我由衷地感谢祖国和母校的培养、感谢导师陈光熙教授和胡明曾教授。

我要感谢北京大学和北京大学深圳研究生院，秉承“兼容并包”的传统，使我能够于2009年底回国后在这个平台上开展科研与学术工作，专心致志、身体力行、知行合一、教学相长。我特别开心的是和研究生们在一起，北大的莘莘学子，求知若渴、思维活跃、风华正茂。

我要感谢澳门科技大学，使我能够延续我的学术生涯。“意诚格物”的校训，勉励我意诚心正、格物致知，激励我继续潜心笃志、脚踏实地。

我要感谢“慕课问道”的布道者——北京大学李晓明教授，使我步入了慕课这个大规模开放式在线课程的平台。从2017年3月起，我的“人工智能原理”慕课课程先后在北京大学“华文慕课”“中国大学MOOC”“北京高校优质课程研究会”以及清华大学“学堂在线”这四个慕课平台上开设。

我要感谢这四个慕课平台，尤其是“中国大学MOOC”，让十几万人工智能的求知者们在网络空间里与这门课程对接。本书就是在我的“人工智能原理”的课堂教学和慕课教学讲义的基础上写成的。当然，讲义和书籍之间有着巨大的差别，我又花了大量的时间重新梳理，许多地方甚至是重写，并充实了大量的内容。

我要感谢*k*-modes等算法的发明人——黄哲学教授，抽出时间审阅这本书，并提出了宝贵的意见。

我要感谢高等教育出版社，使得《人工智能原理》实现了从单薄的讲义到厚实的书籍之蜕变，并得以精心的编辑。

我还要感谢其他几家出版社的约稿，并对我的分身乏术而深表歉意。

我要感谢我的妻子和女儿，她们的理解与支持，使我能够“躲进小楼成一统”，从2009年底回国专心从事科研和学术工作。

最后，我还将此书献给含笑九泉的父母，他们是我回报祖国乃至完成此书的力量之源泉。

王文敏

2019年3月

第二篇

求　解

屈原在《离骚》中有一句名言:“路漫漫其修远兮,吾将上下而求索。”后人的解读是,在追求真知的道路上尽管遥远崎岖,但我将不遗余力地追求和探索。

什么是求解?为什么需要求解?

求解与问题有关。

本书 1.4.1 中介绍了 NP 完全性,涉猎到 NP 完与 NP 难的概念。

对于 NP 完与 NP 难的问题,往往表现为一个巨大的问题空间,用传统的方法难以解决,因此需要“上下而求索”。笔者将屈原的离骚放在第二篇的开场白,其用意仅在于此。

在人工智能领域中,解决上述问题则称其为问题求解(problem solving),简称为求解。斯图尔特·罗素和皮特·诺威在《Artificial Intelligence: A Modern Approach (3rd Edition)》(人工智能:一种现代的方法(第 3 版))一书对此做了详细的论述(Russell,2009),本篇亦参考了该书的一些内容。

在巨大问题空间上进行求解,搜索是一种有效的方法。如果增加一些技巧,则可以缩短搜索的时间或者在有限的时间内得到一个近似解。人工智能的方法,包括启发式、剪枝、蒙特卡罗方法等,都是为了在巨大的问题空间中找到问题的解。

本篇分成四章。笔者在人工智能的问题求解中挑选出四类代表性问题,即搜索问题、优化问题、博弈问题以及约束问题,分别在第 3~6 章论述这四类问题及其求解方法。

第3章　搜索问题求解

PPT：搜索问题求解

3.1 引言

有这样一类问题：具有初始状态和目标状态，每个状态可以看作一个黑盒子，其状态空间能够表示为一棵树（tree）或一张图（graph），这类问题的求解是通过搜索找到一个从初始状态到目标状态的最短路径，而无法用传统的数学方法进行求解。

这类问题称为搜索问题，其求解过程是搜索。

3.2 搜索问题

有很多问题属于搜索问题：一类是人类发明的智力游戏问题（Korf，2012；Levitin，2002），另一类是现实世界的问题。

3.1.1 智力游戏问题

很多智力游戏问题被作为人工智能搜索问题求解的经典实例。例如：八数码难题（8-puzzle）、八皇后难题（eight queens puzzle）、汉诺塔（hanoi tower）、传教士和食人族问题（missionaries and cannibals problems）等。

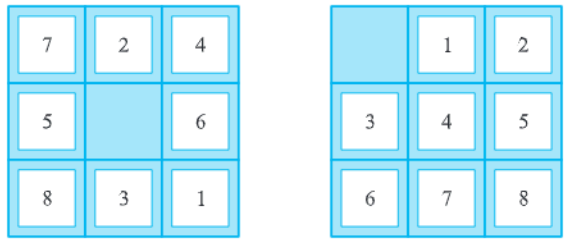
下面介绍其中的三个问题。

例 3.1　八数码难题（8-puzzle）

八数码难题也称为九宫问题、九宫格问题：有一个 3×3 的棋盘，九个格子上摆有八个棋子，每个棋子上标有 1 至 8 中的某个数字，不同棋子上标的数字不相

同。棋盘上余下一个空格，与空格相邻的棋子可以移到空格中。八数码难题中的一个状态就是八个棋子在棋盘上的一种摆法。某个棋子移动后，状态就会发生改变。

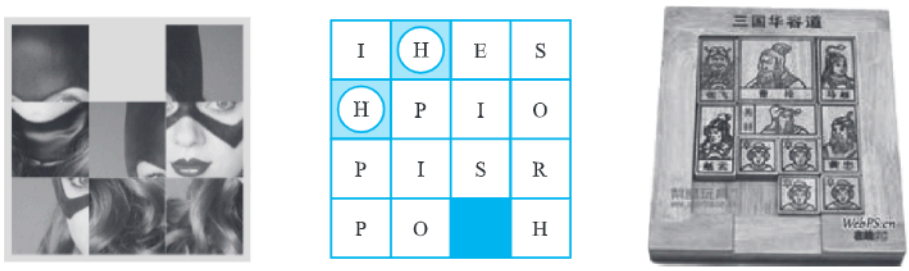
如图 3.1 所示，给定一个初始状态和一个目标状态，目的是找出一种移动步骤，使之从初始状态到目标状态所需移动棋子的步数最少。



(a) 初始状态　　(b) 目标状态

图 3.1　八数码难题举例

八数码难题属于滑块难题(sliding block puzzles)问题。滑块难题的例子有很多，例如：图 3.2(a)是 3×3 滑块难题，表现为一个漫画人物。图 3.2(b)是 4×4 滑块难题，表现形式为英语拼词。而图 3.2(c)则是以“华容道”为题材的滑块难题。



(a)　　(b)　　(c)

图 3.2　三种不同类型的滑块难题

华容道是中国四大经典益智玩具之一，其余的三个是七巧板、鲁班锁以及九连环。而华容道、九连环以及外国的魔方又被誉为世界三大经典益智玩具。

华容道源于三国时期著名的“曹操败走华容道”的故事：三国赤壁大战时，诸葛亮巧用火攻之计大败曹军，且他料到曹操一旦失败必定会从华容道逃走，就派关羽等蜀将守在此地，果然将曹操逮个正着。但关羽终因念及曹操旧恩，放了他一条生路，借此曹操才得以保全性命。

华容道滑块难题有十个棋子:一个曹操,五虎上将——关羽、张飞、赵云、马超、黄忠以及蜀军的四个士卒。有三种棋局,即:横刀立马、层层设防、插翅难逃。要设法用最少的步数把曹操移到出口使其逃走,其他棋子不允许离开棋盘。

滑块难题被认为是 NP 完问题。

滑块难题属于二维组合谜题(2D combination puzzle),还有一类则是三维组合谜题(3D combination puzzle),其代表性的游戏是魔方(Magic cube),中国台湾地区译为魔术方块、香港地区称之为扭计骰。魔方是由匈牙利雕塑家和建筑学教授厄尔诺·鲁比克(Ernö Rubik)于 1974 年发明的,因此,魔方又称为鲁比克方块(Rubik's cube),1980 年起在中国也广为流行。

还有许多其他立体几何形状的组合谜题。

中国民间流传的木制玩具鲁班锁和孔明锁,历史更为悠久。相传鲁班锁是由春秋战国初期鲁国工匠鲁班为了测试儿子的智力,用六根木条制作的一件可拼拆的玩具,后人称其为鲁班锁。而孔明锁则相传是三国时期诸葛孔明根据八卦玄学的原理发明的一种可拼拆的木制玩具。许多人认为鲁班锁和孔明锁是同一类木制玩具,均起源于中国古代建筑首创的榫卯结构。


3-1 组合谜题

二维和三维组合谜题等都属于组合谜题(combination puzzle)。

例 3.2 八皇后难题(eight queens puzzle)

这是国际象棋棋手马克斯·贝瑟尔(Max Bezzel)于 1848 年提出的一个经典而著名的问题:在 8×8 的国际象棋的棋盘上摆放八个国际象棋的皇后,使其不能发生互相攻击的现象,即任意两个皇后都不能处于同一行、同一列或同一斜线上。问有多少种摆法。

图 3.3 是一个八皇后难题的例子:当摆放至第七个皇后时,对角线上有两个皇后,就是说发生了互相攻击的现象。

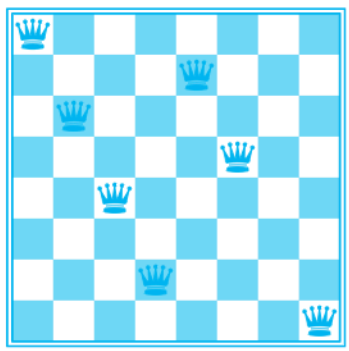


图 3.3 八皇后难题的一个实例

例 3.3 传教士和食人族(missionaries and cannibals)

传教士和食人族是一个经典的智力游戏问题,也是人工智能搜索问题求解的经典问题之一(Pressman,1989)。三个传教士和三个食人族来到一条河的左岸,他们想渡河到右岸。河的左岸有一条小船,只能供两个人乘坐:可以是两个传教士、一个传教士和一个食人族或两个食人族。但是无论在河的左岸或右岸,如果食人族的人数超过传教士,食人族就会把传教士吃掉。问,怎样利用这条船把他们全部摆渡过河。

这个问题推而广之,可以考虑 N 个传教士和 N 个食人族的问题。

3.1.2　现实世界问题

现实世界中有许多问题可以通过搜索进行求解。

例 3.4　最短路径问题(shortest path problem)

给定一个由若干快递网点构成的快递网点布局图(图 3.4),其中的每个节点表示一个快递网点,连接两个节点的边表示快递网点间的路径,边上面的数字表示该条路径的代价。求解其中给定的两个网点之间的最短距离。

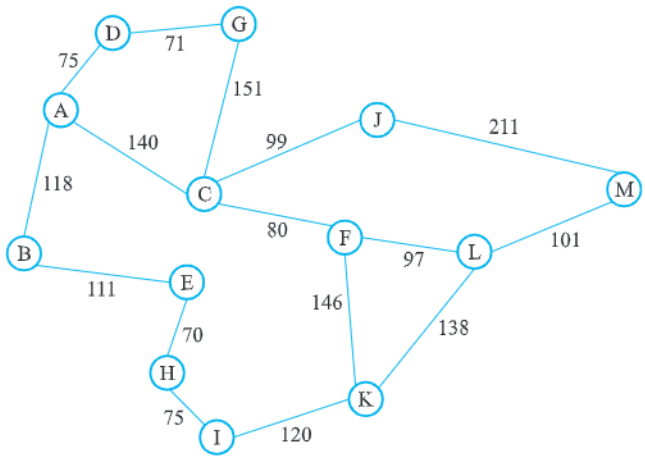


图 3.4　快递网点布局示意图

现实世界中也有很多其他的实例。例如:超大规模集成电路布局(VLSI layout)问题,需要在芯片上以最小化面积、最小化电路延迟、最小化杂散电容等来定位数百万个元件及其连接;此外,还有机器人导航(robot navigation)等。

3.3　搜索问题的要素

搜索问题与状态(state)和状态空间(state space)有关。搜索问题的求解需要对问题进行形式化,其中涉及状态如何表征。

3.3.1　状态表征

状态是问题在不同时期或不同条件下所表现出的形态。问题提出时的状态称其为初始状态(initial state),问题达到预期结果时的状态为目标状态(goal state),其他为中间状态。

搜索问题中状态的表征主要采用原子式表征(atomic representation)。

如图 3.5 所示,所谓原子式表征,指的是每个状态的表征是个黑盒子,不考虑其内部结构。例如,寻找起点至终点间的最短行驶路径问题,所需关心的主要是两点之间的路径而不是每个节点的内部结构,因此,这类问题就可以采用原子式子表征。

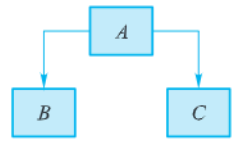


图 3.5 状态的原子式表征

3.3.2 状态空间

定义 3.1 状态空间是问题在不同时期或不同条件下所表现出的所有状态的有机组成。通常可以抽象为一张图(graph)或一棵树(tree)。

① 图 G 由非空有限节点集 N 和边的有限集 E 组成,即:$G=(N,E)$。其中,图中的每个节点表示一个状态,连接两个节点的边表示一个动作,状态空间上的路径(path)则是由一系列动作连接而成的状态序列。

状态空间图可以是一个有向图(directed graph),也可以是一个无向图(undirected graph)。

② 树是图的特例,它是任意两个节点之间连通且没有闭环的图。一棵树只有一个根节点。

3.3.3 形式化

定义 3.2 一个问题,若可以将其表示成状态空间、并且在初始状态和目标状态之间可能存在多条路径的话,就称其为搜索问题(search problem)。

智能主体搜索的目的是从初始节点至目标节点之间找到一条代价(cost)最低的路径。

对于某些 NP 完或者 NP 难问题,将其化为搜索问题是解决该问题的有效途径。

定义 3.3 一个搜索问题可以形式化为一个 5 元组,即:$Search=(S,s_0,A,G_t,c)$。其中:

① S 为问题空间的状态集合。状态 $s\in S$,且 S 不为空。

② s_0表示初始状态,即智能主体开始时的状态。

③ A 表示动作的集合。$s_{i+1}=A_i(s_i)$表示在s_i状态下执行动作A_i,返回新的状态s_{i+1}。而由 n 个动作组成的动作序列则表示为$\{A_1(s_1),A_2(s_2),\cdots,A_n(s_n)\}$。智能主体的动作也可以称为迁移函数(transition function)或后继函数(successor function),就是说:因为动作导致状态迁移,或因动作而生成后继状态。

④ G_t是目标检测(goal test),即判断一个给定的状态是否是目标状态。

⑤ c 表示路径代价(path cost),即该条路径中每一步所需要的代价之和,代价通常表现为一个数值。例如:状态s_i下执行动作A_i到达状态s_{i+1}的代价表示为$c_i(s_i, A_i, s_{i+1})$。

3.3.4 求解的方法

定义 3.4 搜索问题的解是一个从初始状态到达目标状态的路径。搜索问题的求解就是智能主体寻找该路径的动作。

算法 3.1 是一个简单的针对搜索问题进行求解的主体算法。其中有几个有关搜索的重要子函数,即:获取当前状态的 GET-CURRENT-STATE,对目标状态做形式化处理的 FORMULATE-GOAL,对该搜索问题进行形式化的 FORMULATE-PROBLEM,以及完成当前搜索任务的 SEARCH。该智能主体算法中有 5 个局部变量,其输入是感知 *percept*,输出是搜索得到的动作 *action*。

算法 3.1 一个简单的搜索问题求解算法

```
agent SIMPLE-PROBLEM-SOLVING
    input percept
    output an action
    local seq, an action sequence, initially empty
          state, some description of the current world state
          goal, a goal, initally null
          problem, a problem formulation
          action, the most recent action, initially none
    state←GET-CURRENT-STATE(state, percept)
    if seq is empty then
          goal←FORMULATE-GOAL(state)
          problem←FORMULATE-PROBLEM(state, goal)
          seq←SEARCH(problem)
          if seq = failure then return a null action
    action←FIRST(seq)
    seq←REST(seq)
    return action
```

3.4 搜索问题的实例化

这里对 3.2 中介绍的几个经典的搜索问题，按照 3.3.3 给出的形式化定义对其进行具体的形式化操作，这里称其为实例化(instantiation)。

首先对三个经典智力游戏问题，即八数码难题、八皇后难题以及传教士和食人族问题进行形式化，再对一个现实世界的最短路径问题进行同样的处理。

3.4.1 八数码难题

八数码难题是人工智能搜索算法的经典用例。

如前所述，八数码难题有九个格子、八个数字棋子、一个空格。目的是找到一个动作序列，使之从初始状态至目标状态所需移动棋子的步数最少。

① 状态 S：九个格子中八个数字棋子与空格的不同摆放，形成了 $9! = 362,880$个状态。

② 初始状态 s_0：任意一个状态都可以成为初始状态。

③ 动作集合 A：最简单的动作是移动空格，这样只需将空格向左、右、上、下(Left, Right, Up, Down)移动就可以实现状态的迁移。不同的状态依赖于空格的位置。

④ 目标检测 G_t：即检查某个状态是否与目标状态的布局相符。

⑤ 路经代价 c：每一步的代价为 1，故其代价就是路径上的步数。

3.4.2 八皇后难题

八个皇后在 $8\times8=64$ 个格子的棋盘上共有 $C(64,8)=4,426,165,368$ 种摆放方法，但有人已经证明：只有 92 个不同的解，如果将旋转和对称的解归为一种的话，则只有 12 个独立解。

求解八皇后难题有两种形式方法，即：

第一，增量形式化(incremental formulation)：从空状态开始(即棋盘上没有皇后)，然后每个动作添加一个皇后来改变其状态，并使其不和其他皇后发生攻击，直到八个皇后都摆在棋盘上为止。

第二，全态形式化(complete-state formulation)：初始时八个皇后都放在棋盘上(任意摆放)，然后再将这些皇后逐个移开，直到八个皇后不能相互攻击为止。

这里先给出增量形式化的方法。

① 状态集合 S：棋盘上逐次摆放一至八个皇后。

② 初始状态s_0:棋盘上没有皇后。

③ 动作集合 A:每次添加一个皇后至棋盘上任意一个空格,且避免与棋盘上已有的皇后发生攻击。每次得到一个新的状态。

④ 目标检测G_t:八个皇后都已摆放在棋盘上,并且没有攻击。

⑤ 路径代价 c:等于添加皇后的步数。设每步的代价为 1,故其代价是路径上的步数。

八皇后难题已经被推广为更一般的 n 皇后难题,对应的棋盘的大小为 $n\times n$,而皇后个数也是 n。当且仅当 $n=1$ 或 $n\geqslant 4$ 时问题有解(Hoffman,1969)。

3.4.3 传教士和食人族问题

由于传教士与食人族的数目均等于 3,所以只需表示河的左岸的情况,即可算出右岸的结果。这样,左岸的状态可用一个三元组 $S_{\text{left}}=(\text{missionary},\text{cannibal},\text{boat})$来表示,其中 missionary、cannibal 分别代表左岸上传教士与食人族的数目,boat=1 表示船在左岸,boat=0 则表示船不在。

① 状态集合 S:missionary 和 cannibal 都可以取 0~3,boat 为 1 或 0,因此共有 $4\times4\times2=32$ 种状态。

② 初始状态 s_0:传教士、食人族和船都在左岸,即 $s_0=(3,3,1)$。

③ 动作集合 A:用船将传教士和食人族运过河。但要满足问题的约束条件。每次将传教士和食人族运过河后,得到新的状态。

④ 目标检测G_t:传教士、食人族和船都在右岸,即(0,0,0)。

⑤ 路径代价 c:等于路径中的步数。

传教士和食人族问题属于“渡河难题(river crossing puzzles)”,这类问题还有“吃醋丈夫问题(jealous husbands problem)”(Pressman,1989)“火炬过桥问题(bridge and torch problem)”“狐狸、鹅与豆袋难题(fox,goose and bag of beans puzzle)”等。此外,“船夫与羊、狼和白菜渡河问题”亦属于此类。

3.4.4 最短路径问题

对于例 3.4 中的快递网点布局示意图(图 3.4)来说,图中的每个节点表示快递的一个网点。这里采用原子式表征,因为只关注快递网点间的距离。连接两个节点的边表示快递网点间的路径,边上面的数字表示该条路径的代价。

① 状态集合 S:图 3.4 中节点的集合。

② 初始状态s_0:即出发时的状态。设出发地点位于快递网点 A,则初始状态为:$s_0=In(\text{A})$。

③ 动作集合 A:例如$\{Go(\text{B}),Go(\text{C}),Go(\text{D})\}$。

④ 目标检测 G_i：智能主体到达快递网点 M 为目标状态，即：In(K)。

⑤ 路径代价 c：即该条路径每一步所需要的代价之和。

3.5 搜索求解的方式

对搜索问题进行形式化之后，就可以对其通过搜索进行求解。3.3.2 中已经提到，一个搜索问题的状态空间可表示为一张图或一棵树，搜索求解的方法就是图搜索或树搜索。本节先给出树搜索的求解方式，再介绍图搜索。

3.5.1 树搜索

以图 3.6 为例，设起点是 A，终点是 M。下面讨论如何将搜索过程表示为一棵树并通过树搜索方式来求解。

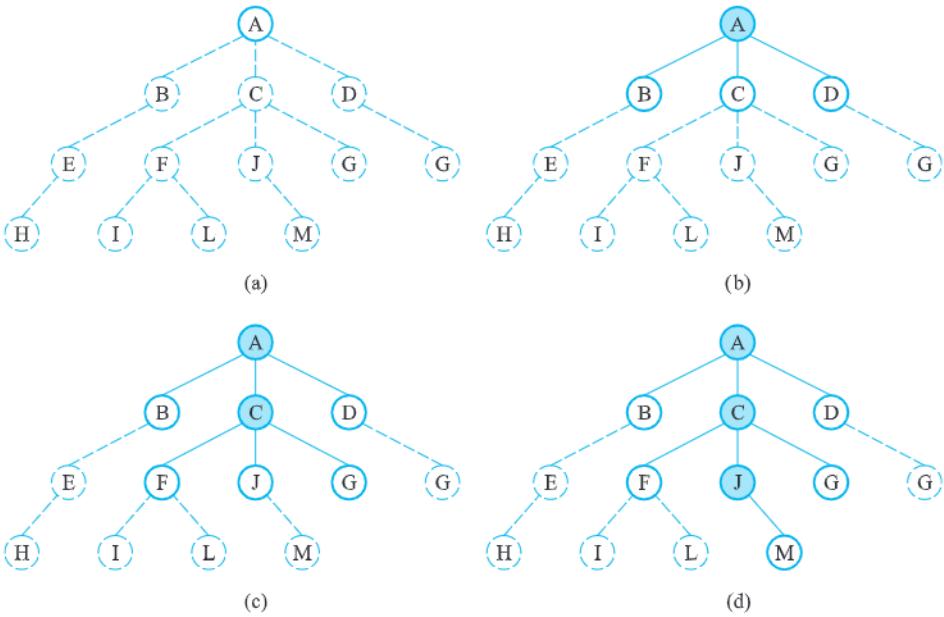


图 3.6 采用树搜索的最短路径问题

将起点 A 作为树的根节点、以终点 M 为树的叶节点。在图 3.6 中，采用粗实线表示为已生成节点、但尚未扩展；虚线表示该节点与路径尚未生成；已被扩展的节点则用粗实线并加上阴影表示。

对起始的根节点 A(图 3.6(a))进行扩展，生成 B、C、D 三个后继节点(图 3.6(b))。

再扩展节点 C,生成 F、J、G 三个后继节点(图 3.6(c))。类似地,再扩展节点 J 后,生成后继节点 M,到达目标节点(图 3.6(d))。

这里给出一个通用的树搜索(tree search)算法,见算法 3.2。

算法 3.2 一个通用的树搜索算法

```
agent TREE-SEARCH
    input problem
    output a solution, or failure
    local frontier, to store the set of all leaf nodes
    initialize the frontier using the initial state of problem
    loop do
        if the frontier is empty then return failure
        choose a leaf node and remove it from the frontier
        if the node contairs a goal state then return the corresponding solution
        expand the chosen node, adding the resulting nodes to the frontier
    loop end
```

其中:*problem* 是一个输入变量;局部变量 *frontier* 是一个数据结构,被称为开放表(open list),用于存储所有的叶节点。

树搜索算法先用 *problem* 的初始状态对开放表 *frontier* 进行初始化,形成树的初始根节点。然后执行如下循环操作:

- 如果 *frontier* 为空(没有节点可扩展)则以 failure 告终;
- 选择一个叶节点并将其从 *frontier* 中移出;
- 如果该节点包含目标状态,则返回对应的解;
- 扩展所选择的节点,并将生成节点添加到 *frontier* 中。

3.5.2 图搜索

算法 3.3 是一个通用的图搜索(graph search)算法。将其与树搜索算法(即算法 3.2)相比较会发现,图搜索算法中多了四行,即带下横线的部分。其中,增加的局部变量 *explored* 也是一个数据结构,称为封闭表(closed list),用于记住每个扩展节点。

算法 3.3 一个通用的图搜索算法

```
agent GRAPH-SEARCH
    input problem
    output a solution, or failure
    local frontier, to store the set of all leaf nodes
          explored, to remember every expanded nodes
    initialize the frontier using the initial state of problem
    initialize the explored to be empty
    loop do
        if the frontier is empty then return failure
        choose a leaf node and remove it from the frontier
        if the node contains a goal state then return the corresponding solution
        add the node to the explored
        expand the chosen node, adding the resulting nodes to the frontier
              only if not in the frontier or the explored
    loop end
```

类似的，图搜索算法也是先用 *problem* 的初始状态对开放表 *frontier* 进行初始化，形成树的初始根节点。再将封闭表 *explored* 初始化为空。然后执行如下循环操作：

- 如果 *frontier* 为空（没有节点可扩展）则以 failure 告终；
- 选择一个叶节点并将其从 *frontier* 中移出；
- 如果该节点包含目标状态，则返回对应的解；
- 将该节点添加到 *explored* 之中；
- 扩展所选择的节点，仅当不在 *frontier* 或 *explored* 之中时，将生成节点添加到 *frontier* 中。

如前所述，具有下横线的语句是图搜索算法中增加的部分。此外，循环操作中最后的语句是直译，目的是与算法 3.3 相对应。

3.6 无信息搜索

定义 3.5 无信息搜索（uninformed search）指的是搜索过程中除了问题本身

之外没有任何其他信息，其过程只关心搜索的步骤，每搜索一步所做的只是生成后继并判断是否是目标节点。

无信息搜索也被称为盲目搜索(blind search)、蛮力搜索(brute-force search)或穷举搜索(exhaustive search)。

无信息搜索依据某种搜索策略遍历后继节点，并依次检查每个后继节点是否为目标节点。搜索策略的区分是依据节点扩展的顺序来定义的。其主要策略是：宽度优先搜索、深度优先搜索、迭代深化搜索、回溯深度优先搜索以及双向搜索等。

无信息搜索也是计算机科学和运筹学的核心问题之一。本节主要介绍宽度优先搜索、深度优先搜索以及迭代深化搜索。回溯搜索将在第 6 章中详细介绍。

3.6.1 宽度优先搜索

宽度优先搜索(breadth-first search)也被翻译成广度优先搜索或先宽搜索。

宽度优先搜索是针对树结构或图结构的遍历搜索算法，其策略是扩展最浅的未扩展节点。所谓最浅，是相对于当前节点而言。

对于树来说，宽度优先搜索从树的根节点开始，优先搜索当前层的所有节点，然后再进入下一层。例如，对于图 3.7，其搜索顺序是节点上所标注的英文小写字母的顺序。对于图而言，则选择任一节点作为根节点。

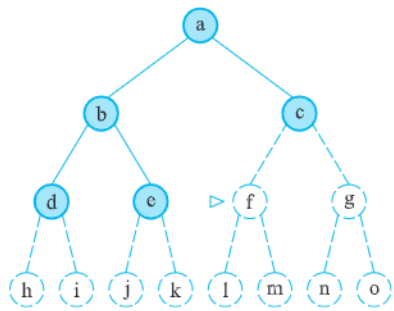


图 3.7 树的宽度优先搜索示意图

实现宽度优先搜索的方法是采用先进先出(first-in first-out，FIFO)队列，即每次将扩展的后继节点放在队列(queue)的后面。

图的宽度优先搜索算法见算法 3.4，其中 *frontier* 作为 FIFO 队列。

算法 3.4 图的宽度优先搜索算法

agent BREADTH-FIRST-SEARCH

input *problem*

output a solution, or failure

local *node*, *frontier*, *explored*

node←a node with State=*problem*.INITIAL-STATE

PATH-TEST=0

frontier←a FIFO queue with *node* as the only element

explored←an empty set

```
loop do
    if EMPTY?(frontier) then return failure
    node←POP(frontier)  //chooses the shallowest node in frontier
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
        child←CHILD-NODE(problem,node,action)
        if child.State is not in explored or frontier then
            if problem.GOAL-TEST(child.State) then return SOLUTION(child)
            frontier←INSERT(child,frontier)
    for end
loop end
```

算法性能评价通常是分析其时间与空间的复杂性。设分枝因子(branching factor)为 b、解的深度为 d,则宽度优先算法的时间复杂性如下:$b+b^2+b^3+\cdots+b^d=O(b^d)$。

对于宽度优先搜索而言,解的深度 d 等于最浅解的深度。空间复杂性亦等于:$O(b^d)$。

对于宽度优先算法而言,当树的宽度和深度很大时,其内存需求及执行时间都成为很大的问题。参考文献(Russell,2009)对此做了定量分析,设:分支因子 $b=10$、每秒钟处理一百万个节点、每个节点占用内存 1 KB,其结果见表 3.1。

表 3.1 宽度优先搜索的时间与空间的定量分析

深度	节点数	时间	内存
2	110	0.11 ms	107 KB (kilobyte)
4	11,110	11 ms	10.6 MB (megabyte)
6	10^6	1.1 s	1 GB (gigabyte)
8	10^8	2 m	103 GB (gigabyte)
10	10^{10}	3 h	10 TB (terabyte)
12	10^{12}	13 d	1 PB (petabyte)
14	10^{14}	3.5 y	99 PB (petabyte)
16	10^{16}	350 y	10 EB (exabyte)

宽度优先搜索的节点数随着树的深度呈现指数级增长,属于指数复杂性问题。下面给出一个指数复杂性问题的经典示例。

例 3.5 汉诺塔(tower of Hanoi)问题

传说印度的婆罗门庙里有 3 根柱子,第一根柱子上有 64 块金盘,要通过第二根

柱子将这些金盘移动到第三根柱子上,如图 3.8 所示。移动的规则很简单:① 每次仅能移动一块金盘,② 仅能移动最上面的金盘,③ 大的金盘不能放在小的金盘上面。寺庙里的祭司一直在移动这些金盘,据说当移动最后一块金盘时,世界将会毁灭!

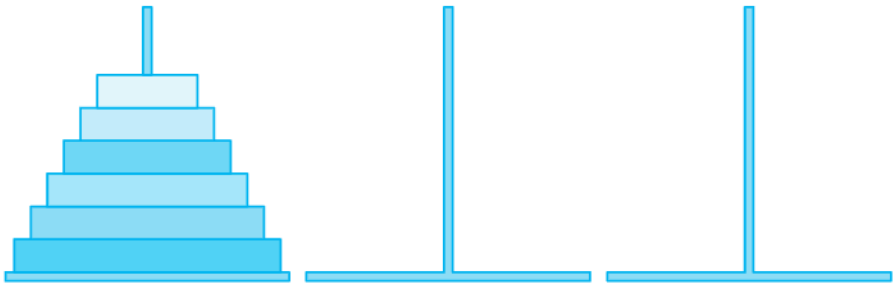


图 3.8 汉诺塔问题

我们估算一下移动金盘所需要的时间。设每秒能移动一块金盘,则移动所有金盘所需时间为 $2^{64}-1$ s,约等于 5,850 亿年!

而地球的天文年龄、即地球开始形成到现在,仅有 45.5 亿年。

3.6.2 深度优先搜索

深度优先搜索(depth-first search)是针对树结构或图结构的遍历搜索算法,其策略是扩展最深的未扩展节点。它与宽度优先搜索的不同之处在于,后者是扩展最浅的未扩展节点。

对于树结构而言,深度优先搜索从根节点开始,沿着一个分枝,一直搜索到叶节点,然后回溯。其搜索顺序可参考图 3.9,即节点标注的英文字母顺序。对于图结构,则选择任一节点作为根节点,直至搜索到图的边界。

实现深度优先搜索的方法是采用后进先出(last-in first-out,LIFO)队列,即每次将扩展的后继节点放在队列的前面。在数据结构(data structure)中,LIFO 队列被称为堆栈(stack)。

设分枝因子为 b、最大深度为 d,则深度优先算法的时间复杂性为:$O(b^d)$。

对于树结构而言,深度优先搜索算法只需保存从根节点到叶节点的单个路径以及该路径上每个尚未扩展的剩余节点。一旦某个叶节点已被检测为非目标节点,则可以将其从内存中删除。因此,树结构深度优先搜索的特点是其空间复杂性低于宽度优先搜索。

如图 3.9(a)所示,其中节点 c 扩展后生成两个叶节点 d 和 e,设这两个叶节点都不是目标节点,将它们从内存中删除,形成图 3.9(b)。此时 c 已成为叶节点,再将其删除,结果变为图 3.9(c)。以此类推。

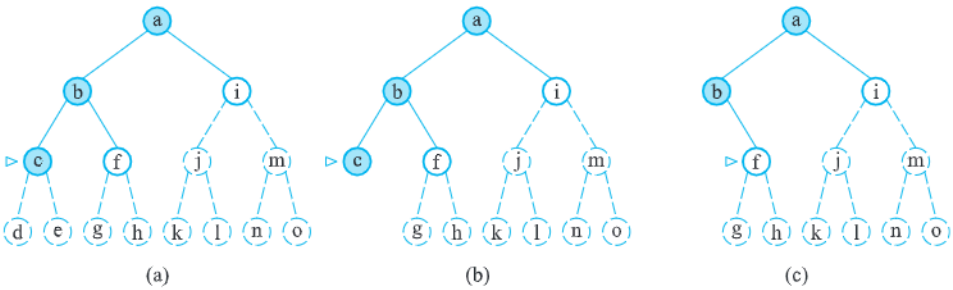


图 3.9 深度优先搜索算法示例

因此,树结构深度优先搜索的空间复杂性是:$O(bd)$。

对于图结构的深度优先搜索,则没有上述优点。

3.6.3 迭代深化搜索

对于深度优先搜索算法来说,如果遇到一棵深度无限的树,就会沿着某一侧的分枝一直搜索下去。这个问题可以用预先设定的深度限制 l 加以解决,位于深度 l 的节点被视为叶节点。这种方法被称为深度受限搜索(depth-limited search),也被称为深度受限深度优先搜索(deep-limited depth-first search)。

设目标节点的深度为 d。若 $l<d$,即目标节点位于深度限制 l 以外,深度受限搜索则找不到目标。因此,深度限制 l 的选择变得非常重要。但是,目标节点的深度往往在搜索之前是不得而知的。

迭代深化搜索(iterative deepening search)可以解决上述问题,它将深度优先和宽度优先的优势相结合,逐步增加深度限制 l,直到找到目标为止。在每次迭代内部,它以深度优先搜索的顺序遍历搜索树的节点;但在迭代之间,其顺序相当于宽度优先。迭代深化搜索也被称为迭代深化深度优先搜索(iterative deepening depth-first search)。

迭代深化搜索算法见算法 3.5。其中,深度限制 *limit* 从 1 开始调用DEPTH-LIMITED-SEARCH 函数,而该函数是一个递归函数(recursive function),在其内部完成深度受限搜索。

算法 3.5 迭代深化搜索算法

```
agent ITERATIVE-DEEPENING-SEARCH
    input problem
    output a solution, or failure
    root = MAKE-NODE(problem.INITIAL-STATE)
```

```
    for limit = 1 to ∞ do
        result ← DEPTH-LIMITED-SEARCH(root, problem, limit)
        if result ≠ cutoff then output result
    end for
function DEPTH-LIMITED-SEARCH(node, problem, limit)
    if problem.Goal-Test(node.State) then return Solution(node)
    if limit = 0 then return cutoff   //no solution
    cutoff_occurred ← false
    for each action in problem.ACTIONS(node.STATE) do
        child ← CHILD-NODE(problem, node, action)
        result ← DEPTH-LIMITED-SEARCH(child, problem, limit-1)
        if result = cutoff then cutoff_occurred ← true
        else if result ≠ failure then return result
    end for
    if cutoff_occurred then return cutoff   //no solution
    else return failure
```

3.7 有信息搜索

定义 3.6 有信息搜索(informed search),是在搜索过程中依据问题提供的特有信息、或动态生成的有用信息,用以引导其搜索策略沿着有希望的搜索路径尽快找到目标。

与无信息搜索不同,有信息搜索生成一个评价函数(evaluation function),用于搜索的代价估计(cost estimate),具有最低代价的节点被优先扩展。

评价函数记作 $f(n)$,表现为如下形式:

$$f(n)=g(n)+h(n) \tag{3-1}$$

其中,$g(n)$是从初始节点至节点 n 之间的路径代价(path cost),而 $h(n)$则是节点 n 到目标节点之间路径的最低估计代价,称为启发式函数(heuristic function)。

从评价函数可以看出,有信息搜索策略可以分为如下三种情况:

① 仅考虑路径代价,即:$f(n)=g(n)$;

② 只依据启发式函数,即:$f(n)=h(n)$;

③ 兼顾路径代价与启发式函数,即:$f(n)=g(n)+h(n)$。

有信息搜索可看做是最佳优先搜索(best-first search),就是说,这类搜索算法依据所得到的信息,按既定的"最佳优先"策略进行搜索。

有信息搜索包括:迪杰斯特拉(Dijkstra)算法、统一代价搜索、贪婪搜索、A*搜索以及迭代深化 A*搜索等。这些算法的共同特点是根据评价函数 $f(n)$ 来决定选择哪一条路径,最好的 $f(n)$ 值作为优先选择路径。

本节主要介绍统一代价搜索、贪婪搜索、A*搜索。而迭代深化 A*搜索是3.6.3介绍的迭代深化搜索与 A*搜索的结合,在此不做详述。

3.7.1 统一代价搜索

统一代价搜索(uniform cost search)是一种树搜索算法,它在扩展节点时根据问题提供的每段路径代价,寻找一条从起始节点至目标的代价最低的路径,就是说,统一代价策略仅考虑路径代价,其评价函数为 $f(n)=g(n)$。统一代价搜索算法见算法 3.6。

算法 3.6 统一代价搜索算法

```
agent UNIFORM-COST-SEARCH
    input problem
    output a solution, or failure
    local node←a node with STATE=problem.INITIAL-STATE, PATH-TEST=0
          frontier←a priority queue ordered by PATH-COST, with node as the
           only element
          explored←an empty set
    loop do
        if EMPTY(frontier) then return failure
        node←POP(frontier)   //chooses the lowest-cost node in frontier
        if problem.GOAL-TEST(node.State) then return Solution(node)
        add node.STATE to explored
        for each action in problem.ACTIONS(node.STATE) do
                child←CHILD-NODE(problem, node, action)
                if child.STATE is not in explored or frontier then
                        frontier←INSERT(child, frontier)
        else if child.STATE is in frontier with higher PATH-COST then
            replace that frontier node with child
```

```
        end for
    end loop
```

设当前节点 $n=n_k$，初始节点为 n_0，两点之间某条路径上的所有节点为 $\{n_0, n_1, \cdots, n_k\}$，两点之间路径代价（path cost，PS）记做 $g_{PS}(n_i, n_{i+1})$，则该条路径的代价 $g(n)$ 为

$$g(n) = g_{PS}(n_0, n_1) + \cdots + g_{PS}(n_{k-1}, n_k) = \sum_{i=0}^{k-1} g_{PS}(n_i, n_{i+1}) \qquad (3-2)$$

为了便于说明问题，这里举一个例子：将图 3.4 简化为图 3.10，目的是寻找节点 C 至节点 M 之间代价最低的路径。

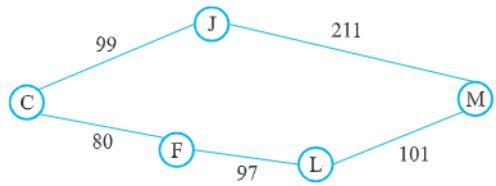


图 3.10 快递网点的子图

从 C 出发，至后继节点 F 的路径代价 $g_{PS}(C,F)=80$ 最低，扩展该节点得到其后继节点 L，这条路径的代价是：

$$g_{PS}(C,F)+g_{PS}(F,L)=80+97=177$$

此时，代价最低的路径变成 $g_{PS}(C,J)=99$，扩展该节点得到目标节点 M，其路径代价：

$$g_{PS}(C,J)+g_{PS}(J,M)=99+211=310$$

接着比较另一条路径的代价。扩展 L 后也得到目标节点 M，路径代价是：

$$g_{PS}(C,F)+g_{PS}(F,L)+g_{PS}(L,M)=177+101=278$$

相比之下，$278<310$，故经由 $\{C,F,L,M\}$ 的路径代价最低。

3.7.2 贪婪最佳优先搜索

贪婪最佳优先搜索（greedy best-first search）的策略是试图扩展最接近目标的节点，这也是贪婪名称的由来。其搜索策略是有信息搜索中的第二种，$f(n)=h(n)$，即仅使用启发式函数 $h(n)$ 对节点进行评价，而该启发式函数是一种贪婪的策略，即从节点 n 到目标的最低估计代价。

这里考虑例 3.4 的快递网点最短路径问题，设当前节点 $n=n_k$，目标节点为 n_g，则将这两点之间的直线距离（straight line distance，SLD）作为启发式函数，记做：$h(n)=h_{SLD}(n_k,n_g)$。图 3.4 中的各节点到目标节点 M 之间的直线距离详见表 3.2。

表 3.2 各节点到目标节点 M 之间的直线距离

$h_{SLD}(A,M)$	366	$h_{SLD}(H,M)$	241
$h_{SLD}(B,M)$	329	$h_{SLD}(I,M)$	242
$h_{SLD}(C,M)$	253	$h_{SLD}(J,M)$	176
$h_{SLD}(D,M)$	374	$h_{SLD}(K,M)$	160
$h_{SLD}(E,M)$	224	$h_{SLD}(L,M)$	100
$h_{SLD}(F,M)$	193	$h_{SLD}(M,M)$	0
$h_{SLD}(G,M)$	380		

下面,依据表 3.2 给出的数据作为 $h_{SLD}(n_k,n_g)$,来求解例 3.4 中从初始节点 A 到目标节点 M 的路径。

如图 3.11 所示,首先对起始节点 A 进行扩展,得到后继节点 B、C 和 D。由表 3.2 可知,这三个节点到达目标节点 M 的直线距离分别为 329、253 和 374。其中节点 C 到达 M 的直线距离最小,扩展 C 得到后继 F、G 和 J。类似的,节点 J 到达 M 的直线距离最小,扩展 J 得到目标节点 M。

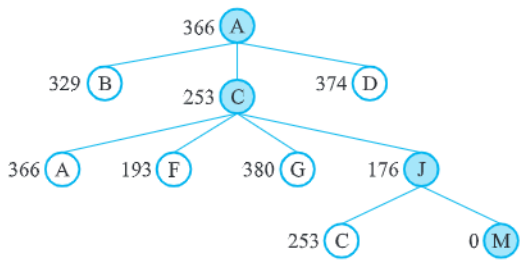


图 3.11 贪婪最佳优先搜索求解

因为贪婪搜索的评价函数是 $f(n)=h(n)$,就是说,评价函数仅使用启发式函数。对这个具体问题来说,用当前节点至目标之间的直线最短距离 $h_{SLD}(n_k,n_g)$ 作为启发式来找到解答,因此该启发式搜索的代价是最小的,然而它不是全局最优的。对此,可以通过计算路径代价来说明为什么不是全局最优。

第一条路径是从初始节点 A 出发,经由 C 和 J,再到达目标节点 M。其路径代价如下:

$$g_{PS}(A,C)+g_{PS}(C,J)+g_{PS}(J,M)=140+99+211=450$$

第二条路径是从初始节点 A 出发,经由 C、F 和 L,再到达目标节点 M。其路径代价是:

$$g_{PS}(A,C)+g_{PS}(C,F)+g_{PS}(F,L)+g_{PS}(L,M)=140+80+97+101=418$$

显而易见,第一条路径要比第二条远 32 公里。这一点恰恰说明为什么称这种搜索方式是“贪婪”,因为每一步它都试图得到能够最接近目标的节点,但并非是全局最优的。

3.7.3　A* 搜索

A* 搜索(A-star search)的策略是路径代价与启发式函数这两者兼顾的搜索,可以看作是统一代价优先和贪婪最佳优先搜索这两者结合的产物。

这里仍然针对例 3.4 的快递网点最短路径问题,采用 A* 搜索进行求解,各节点到目标节点 M 之间的直线距离如表 3.2 所示。其评价函数是:$f(n)=g(n)+h(n)$。

图 3.12 中每个节点旁边标注的是采用上述评价函数并代入相应的数据后得到的结果。例如,对于节点 F,路径为{A,C,F},因此,$g(n)=g_{PS}(A,C)+g_{PS}(C,F)=140+80=220$,$h(n)=h_{SLD}(F,M)=193$,故 $f(n)=g(n)+h(n)=220+193=413$。

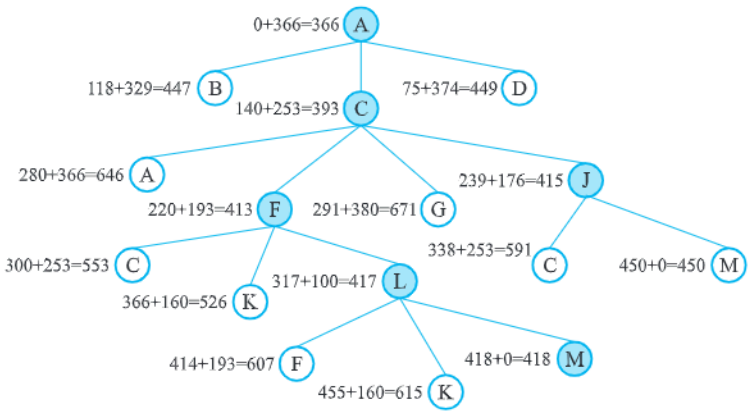


图 3.12　A* 搜索求解

在求解过程中,节点 F 扩展生成的三个后继 C、K 和 L 的值为 553、526 和 417,大于节点 J 的值 415。因此对节点 J 进行扩展,生成的后继 C、M 的值为 591、450,其中 M 的值大于节点 L 的值。这样再对 L 进行扩展,生成的后继 M 的值为 418。故途经节点{A,C,F,L,M}的路径为最终的解。

A* 搜索具有如下性质:若启发式函数 $h(n)$ 是可接受的(admissible),则 A* 的树搜索版(tree-search version)是最优的;若 $h(n)$ 是一致的(consistent),则 A* 的图搜索版(graph-search version)是最优的(Russell,2009)。

3.8 小结

本章在介绍几种智力游戏问题和现实世界的问题之后，论述了搜索问题的要素及其形式化方法。

搜索空间可以表示为树或图的结构，与其对应的搜索求解方式就是树搜索和图搜索。

根据搜索过程中是否具有问题提供的特定信息来生成评价函数，又分为无信息搜索和有信息搜索。

无信息搜索也被称为盲目搜索、蛮力搜索或穷举搜索。代表性的搜索策略包括宽度优先、深度优先以及迭代深化搜索。

有信息搜索可看做是最佳优先搜索，包括统一代价搜索、贪婪搜索、A* 搜索以及迭代深化 A* 搜索等。

习　题

3.1　为什么某些问题只能通过搜索进行求解？什么类型的问题可作为搜索问题？

3.2　请给出一个本书之外的搜索问题的例子，智力游戏或现实世界的例子均可。

3.3　为什么搜索问题的状态采取原子式表征？是否还需要其他的方式？为什么？

3.4　搜索问题的状态空间可抽象为一张图或一棵树，图和树有什么区别？

3.5　搜索问题的形式化有什么作用？

3.6　叙述树搜索算法与图搜索算法的相同与不同之处。

3.7　拉斯维加斯算法（Las Vegas algorithm）是一种随机化算法（randomized algorithm）。请查阅相关文献，并了解该算法求解八皇后难题的步骤。并讨论该方法与本章讲述的求解八皇后难题的两种形式化方法的区别。

3.8　论述无信息搜索和有信息搜索之间的相同与不同之处。

3.9　有信息搜索的评价函数 $f(n)=g(n)+h(n)$ 的含义是什么？可分为几种情况？

3.10　能否找到一个通用的、适用于所有搜索问题的启发式函数 $h(n)$？为什么？