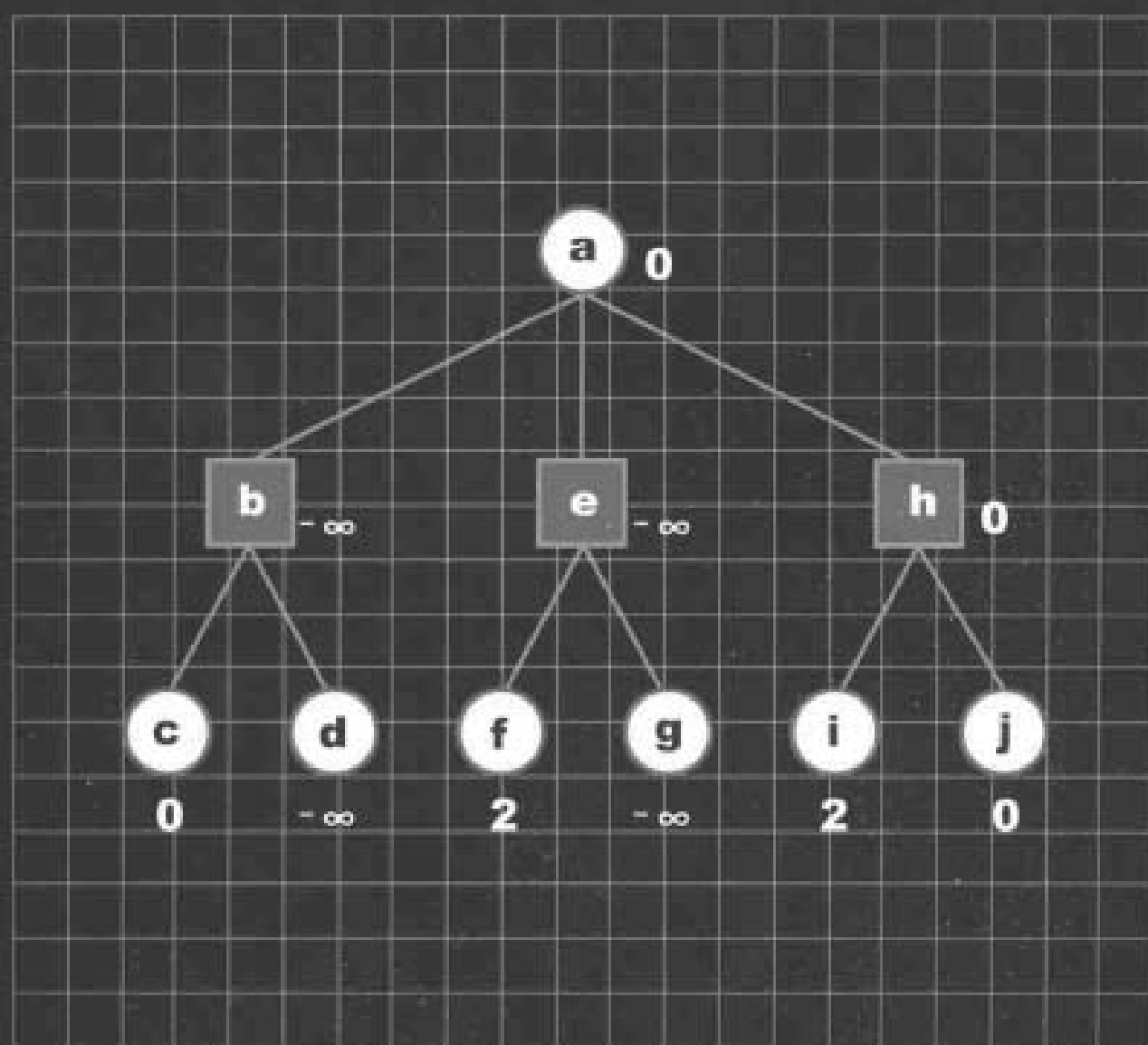


OHM 大学理工系列

人工智能

〔日〕沟口理一郎 石田 亨 编



科学出版社

www.sciencep.com

(TN-0426.0101)

责任编辑 崔炳哲 樊友民

责任制作 魏 谨

责任印制 刘士平

封面设计 李 力

本书内容简介

本书是“OHM大学理工系列”之一。书中简明扼要地介绍了基于搜索的问题求解、知识表示和推理、机器学习、模糊理论-神经网络-遗传算法、模糊识别,以及人工智能语言等。本书讲解条理清晰、通俗易懂,内容基本涵盖了人工智能学科的基础知识,选材上既注意了知识的系统性,也考虑了信息工程相关学科学生学习人工智能的需求。

本书可作为大学相关专业本科生的教材,也可作为相关专业技术人员及研究人员的参考用书;还可供信息工程相关学科的师生、技术人员及研究人员参考。

OHM 大学理工系列

系统软件工程

能源环境学

机电一体化

光与电磁波工程

半导体器件

激光工程

电化学

电磁学

集成电路

电力系统工程

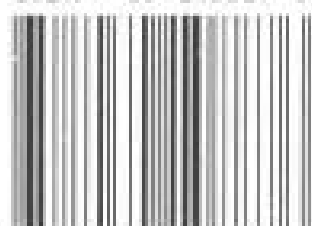
人工智能

光电子学

计算机图形学

网络技术原理及其应用

ISBN 7-03-010887-6



9 787030 108876 >

ISBN 7-03-010887-6

定价: 25.00 元

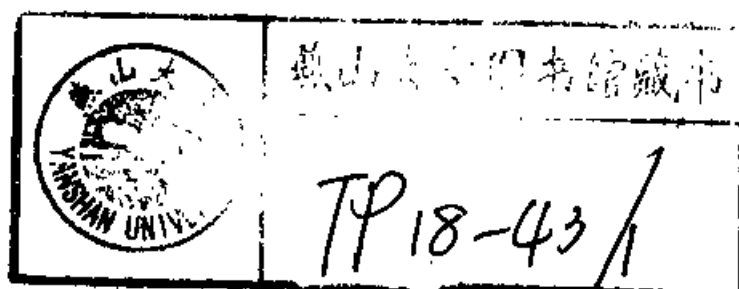
TP



111.14/57

人 工 智 能

〔日〕沟口理一郎 石田 亨 编
卢伯英 译



科学出版社
北 京



0402771

-75

05
10
02

图字:01-2002-2262 号

Original Japanese language edition

Shinseidai Kougaku Series: Jinkou Chinou

Edited by Riechirou Mizoguchi, Tooru Ishida. Written by Yasuhiko Kitamura, Seiechirou Dan, Shigeyoshi Tsutsui, Hirohide Haga, Hitoshi Ogawa, Isao Hayashi and Youichi Yamashita

Copyright © 2000 by Riechirou Mizoguchi and Tooru Ishida

Published by Ohmsha, Ltd.

This Chinese version published by Science Press, Beijing

Under license from Ohmsha, Ltd.

Copyright © 2002

All rights reserved

新世代工学シリーズ

人工知能

溝口理一郎 石田 亨 オーム社 2000

图书在版编目(CIP)数据

人工智能/(日)沟口理一郎,石田 亨编;卢伯英译. —北京:科学出版社,2003

(OHIM 大学理工系列)

ISBN 7-03-010887-6

I. 人… II. ①沟…②石…③卢… III. 人工智能-高等学校-教材 IV. TP18

中国版本图书馆 CIP 数据核字(2002)第 072011 号

责任编辑 崔炳哲 樊友民 **责任制作** 魏 谨

责任印制 刘士平 **封面设计** 李 力

科学出版社 出版

北京东黄城根北街 16 号 邮政编码:100717

<http://www.sciencep.com>

源海印刷厂 印刷

北京东方科龙图文有限公司 制作

<http://www.okbook.com.cn>

科学出版社发行 各地新华书店经销

2003 年 2 月第 一 版 开本: B5(720×1000)

2003 年 2 月第一次印刷 印张: 12

印数: 1 5 000 字数: 144 000

定 价: 25.00 元

(如有印装质量问题,我社负责调换〈新欣〉)

丛书序

主编 樱井良文

现在,很多大学正在进行院系调整以及学科、专业的重组,以研究生培养为重点,引入学期制,采用新的课程体系授课,特别是由于学期制教学计划的引入,使得原来分册编写的教材很难在一个学期的教学中消化。因此,各学校对“易教”、“易学”教材的需求越来越迫切。

本系列是面向通信、信息,电子、材料,电力、能源,以及系统、控制等多学科领域的新型教学参考系列。系列中的各册均由活跃在相应学科领域第一线的教授任主编,由年轻有为的学者执笔,内容丰富、精炼,有利于对学科基础的理解。设计版面时着重为学生留出了写笔记的空间,是一种可以兼作笔记,风格别致的教学参考书。

希望肩负新世纪工程技术领域发展重任的青年读者们,通过本教程系列的学习,建立扎实的学科基础,在实践中充分发挥自己的应用能力。

OHM 大学理工系列编辑委员会

主 编

樱井良文 大阪大学名誉教授

副主编

西川祐一 大阪工业大学校长
京都大学名誉教授

编委(按姓氏笔画顺序)

广濑全孝	广岛大学教授	井口征士	大阪大学教授
木村磐根	大阪工业大学教授 京都大学名誉教授	仁田旦三	东京大学教授
白井良明	大阪大学教授	西原 浩	大阪大学教授
池田克夫	京都大学教授	滨川圭弘	立命馆大学副校长 大阪大学名誉教授

前 言

近年来,虽然出版了一些有关人工智能的优秀著作,但还是不断听到人们抱怨,因为至今还缺少一种适合于大学工科专业半年用的、相当于两个学分的教科书。这种教科书既要简明扼要,又要能涵盖本学科的要点。可能是由于人工智能的研究历史比较短,研究对象涉及的范围又非常广泛,再加上阐明人工智能的学科理论尚未完全确立等原因,造成了目前这种状况。

基于对这种状况的认识,编者在编写这部适合于半年两学分的人工智能入门教材时,在内容安排方面进行了认真考虑,对造成目前状况的原因中,一些有可能得到解决的问题,认真地研究了它们的解决方法。

编写本书时,首先要做的事是确定基本观点。这里存在着两种不同的要求:一种是从进行人工智能研究的观点出发,它着眼于培养未来的人工智能研究新人,因此要求本书在选材上,应注重对他们进行系统完善的知识培养;另一种观点则认为在选材上,不仅要考虑到对与信息工程相关学科的学生有用,而且还要考虑到对那些需要应用人工智能的其他学科的学生也有用。但是在有限的篇幅内,要作到两全齐美是困难的。因此,根据本系列丛书以及本教科书的编写宗旨,采纳了第二种观点进行编写。

其次要解决的问题是确定要编写的内容。除了概述一章外,共设置六章,涉及六个课题,分别是通过搜索解决问题、知识表示与推理、机器学习、模糊理论-神经网络-遗传算法、模式识别,以及人工智能的语言。编者认为,这些内容是人工智能研究中的基础知识,而且对于应用人工智能技术的学生来说,也是必要而且充分的一些课题。再次是关于执笔者的选定问题。这时有两种方案可供选择,一种方案是把这一题材广泛的教材,托付给一位执笔者去完成,另一种方案是针对各个课题,聘请精通该课题的专家分头编写,本书采用了后一种方案。这种作法,可以实现对每一个课题都进行恰如其分地描述,但是同时,在各课题的描述风格和子课题的选择等方面,也可能导致不一致,然而由于编者负有统编全书的

责任,所以这种因分头编写产生的问题,会得到合理解决。实际上,编者聘请了七位富有朝气的学者作为本书的执笔者,他们都是所承担课题方面的专家,而且两位编者独立地对全书草稿仔细阅读以后,进行了统一批注,然后在与执笔者进行详细讨论后,对内容进行了修改,通过对全书的修订,尽力保证了全书的一致性。

各章在技术和理论的阐述过程中,加进了一些热门话题,并且还以加边框的形式,插入了一些学术方面的最新动向等信息,以设法引起读者的兴趣。此外,在每章的最后还编排了一些练习题,并在书末一并给出了练习题的简要解答。另外,还介绍了一些经过精心挑选的文献,可以作为读者进一步学习时的参考,这些文献也适合于学生们进行自学。

本书就是通过上述运作完成的。编者和执笔者通过共同努力,竭尽全力希望把本书编写得最好,但是对本书的最终评价,当然还是要由读者来进行。本书可以用来作为学生研究人工智能的入门书籍,如果本书能对读者有所帮助,编者将感到荣幸。

编 者

目 录

第 1 章	人工智能概述	1
1.1	什么是人工智能	1
1.2	人工智能的历史	5
第 2 章	基于搜索的问题求解	9
2.1	搜索与人工智能的关系	9
2.1.1	八数码魔方	9
2.1.2	状态空间表示	10
2.1.3	与图有关的术语	11
2.2	逐个搜索	12
2.2.1	随机搜索	12
2.2.2	CLOSED 表的引入	12
2.2.3	OPEN 表的引入	13
2.2.4	纵向搜索	14
2.2.5	横向搜索	15
2.2.6	均一代价搜索	17
2.3	应用智能的搜索	19
2.3.1	启发式搜索	19
2.3.2	登山法和最佳优先搜索	20
2.3.3	A* 算法	20
2.3.4	约束的利用	23
2.4	对问题进行分割后进行搜索	23
2.4.1	与/或(AND/OR)图表示	23
2.4.2	与/或(AND/OR)图搜索	26
2.5	博弈树的搜索	27
	练习题	31
第 3 章	知识表示和推理	33
3.1	知识与推理中的关系	33
3.2	产生式系统	34

3.2.1	产生式系统的构造	34
3.2.2	推理机构的运行	37
3.2.3	理由(Why)和方法(How)	41
3.2.4	产生式系统的特征	42
3.3	框 架	42
3.3.1	典型知识与框架	43
3.3.2	阶层知识与特征的继承	45
3.3.3	程序知识及其启动	47
3.3.4	框架的特征	47
	练习题	48
第 4 章	机器学习	49
4.1	关于学习和机器学习	49
4.1.1	什么是学习	49
4.1.2	机器学习的研究历史	51
4.1.3	机器学习的分类标准	52
4.2	应用归纳方法由示例学习概念的定义	55
4.2.1	温斯顿的拱学习	55
4.2.2	决策树的学习	60
4.3	根据丰富的知识和经验提高推理效率	68
4.3.1	效率化学习	68
4.3.2	基于解释的学习(EBL)	69
	练习题	73
第 5 章	模糊理论-神经网络-遗传算法	75
5.1	模糊理论	75
5.1.1	什么是模糊理论	75
5.1.2	模糊集合与普通集合的 区别	76
5.1.3	模糊数也是数吗?	78
5.1.4	模糊控制是一种方便的控制 方法	80
5.2	神经网络	85
5.2.1	什么是神经网络	85
5.2.2	神经元及其学习功能的研究	85
5.2.3	误差反向传播学习是一种便利	

方法	89
5.2.2 神经元及其学习功能的研究	85
5.3 遗传算法	94
5.3.1 什么是遗传算法	94
5.3.2 单纯 GA 的基本步骤	96
5.3.3 简单函数最优化举例	98
5.3.4 单纯 GA 的扩张	100
5.3.5 模式定理	103
5.3.6 遗传算法的应用	104
5.3.7 遗传算法的一些同类方法	106
练习题	106
 第 6 章 模式识别	 109
6.1 什么是模式识别	109
6.2 模式的特征	110
6.3 根据特征模式匹配进行识别	111
6.3.1 用一个参考模式代表类	111
6.3.2 用多个参考模式代表类	114
6.4 基于统计决策理论的识别	116
6.5 对声音的识别	117
6.5.1 根据与参考模式的匹配识别 单词	118
6.5.2 基于统计决策理论的单词识别 ...	122
6.5.3 基于统计决策理论的连续声 音识别	125
练习题	127
 第 7 章 人工智能语言	 129
7.1 人工智能语言是怎样一种语言	129
7.2 函数型语言 Lisp	133
7.2.1 表:具有递归结构的数据	133
7.2.2 Lisp 程序的基本结构	134
7.2.3 由程序到数据和由数据到 程序:eval 和 quote	136
7.2.4 表操作	138

7.2.5 其他的 Lisp 函数	142
7.2.6 Lisp 的执行例子	143
7.3 逻辑型语言 Prolog	148
7.3.1 项:具有递归结构的另一种 数据结构	148
7.3.2 逻辑型语言的计算方法: 归结原理	150
7.3.3 Prolog 的对象:Horn 逻辑式	152
7.3.4 Prolog 程序的表示法	154
7.3.5 单一化(unification)	155
7.3.6 Prolog 的表处理	158
7.3.7 Prolog 的执行控制功能:自动回溯 和截断符号	159
7.3.8 把程序变成数据,把数据变成 程序:assert,retract 及 univ	162
7.3.9 其他的谓词	164
7.3.10 Prolog 的执行例子	164
练习题	168
 练习题简答	 169
参考文献	175

第 1 章 人工智能概述

1.1 什么是人工智能

人们能学习、说话和思考,并且能付诸行动;通过对话和行动,可以与环境(也包含其他人)相互作用,从而可以进一步学习、思考并不断成长。在这些方面虽然人与其他动物有共同之处,但是无论在哪一方面,人类的能力都远远处于动物之上。概括这种情况,称人是“有智慧的”。在计算机诞生的同时,人的这种智慧能力,即“智能”,就开始被列为研究对象。运用计算机,以人类智慧的机理和实现作为研究目标的工作,称为“人工智能”。

进行人工智能研究,也许不是一件容易的事情,但却是一件新奇的事情。人的智慧能力,充满了神秘的魅力,从而引起了人们探索智能机理的强烈欲望。不言而喻,人具有出色的语言功能、视觉功能和听觉功能等,这些都有赖于人的学习功能。运动能力同样是这样,也还是依赖于这种学习机制。特别是掌握语言的能力,这是人们特别感兴趣的。儿童在出生一两年后即开始说话。如果在幼儿时期从一个国家到了另一个国家,同样在一两年内就可以学会所在国的语言。而且由于人们具有记忆这种本能,所以人们会具有丰富的知识和经验,人们的记忆,就是人们的记忆(回忆起必要的事情)能力。记忆能力被认为是构成人的智慧能力的基础。经过一年后再回想人的名字等会变得很困难,但是必要时,通常会及时地回忆起一些必要的知识和发生的事件。这并不难理解。进一步讲,受过特别教育和训练的人,具有出色的解决问题的能力(如医生对疾病的诊断,设计师对汽车的设计等)。考虑到本书的大多数读者都经历过严格的应试奋斗历程,所以他们不仅能够记

忆起以往求解试题的体验,而且能够运用被称为“推理”的求解试题的知识(公式和解法等)进行解题。构成推理,即所谓“思考”的基础是智慧机能。计划的设计和随机应变的实施等,这些富有智慧的行为不胜枚举。

以上谈到的是关于个人的智慧行为,人作为群体,也能构成智慧行为。在组织进行足球、篮球等比赛时,人们看到的组织策划及公司进行的一些运作等,就是一些协调工作(行动)。在这些协调行动中,全体人员具有共同的目标,各个成员一方面根据预先大致确定的各自承担的任务,实施动态的最优化,另一方面要采取一些独立的行动,其结果是实现作为组织(团队)整体的“智慧”行动。

这里我们举一个例子,以便揭示出智能处理中包含的具体内容。当人们经常无意识地对自己的行为进行反省时,人们智能的魅力就会充分地显示出来。下面是A先生和B先生在地铁列车内,关于他们熟悉的C先生的一段谈话。

A:C先生呀,就是在走廊阅读那本“哥德,埃希亚,巴赫”^[1]的原著的那位!

B:哎呀!那个人真了不起,在走廊里读书,是个怪才。

A:是的。因为那本书有700页,所以每天的阅读速度达到70页。

B:是的……(哎呀,是用了10天啊!我把“10日”听成“走廊”了。)

谈话也许就这样结束了吧。在电气列车的噪声干扰下,B先生把10日(とおか)误听为走廊(ろうか)了¹⁾。因为B先生知道C先生是一位优秀的但性情古怪的人,所以,如果说忙碌的C先生站在走廊下进行认真地阅读,在谈话过程中,人们很少怀疑会有其他“可能的解释”。但是,从A先生后面的谈话可以得到下列“新信息”,即“以每一天70页的速度阅读一本700页的书”,据此进行除法运算,可以算出读完该书需要10天时间。一旦意识到这个计算结果,B先生就能够想到最初听到的“ろうか”(走廊)其实是“とおか”(10日)。

这种分析方法虽然有点过分依赖听觉,但是对于人类社会来说,这种处理还是容易进行的,但是对于智能来说,我们认为它没有太大意义。不过人们希望能利用这种处理,开发出语音识别系

1) 日语“とおか”(tookaka)(10日)与“ろうか”(rooka)(走廊)的发音相近。 译者注

统,从而创造出人间奇迹。如果在某些方面,我们对新颖(困难)之处再稍微详细地作一些探讨,那么显然,下列一些问题将成为必须处理的关键性问题:

(1) **语音识别处理** 语音识别处理如同在第 6 章中所述,是先将语音信号数字化,然后进行特征提取,再根据设想的识别对象(如单词等)进行预先准备好的识别处理。在识别处理中,有时也采用第 5 章中介绍的神经网络技术。由于多年来的研究结果,可以说识别技术已经有很大进步,但是在单词(音韵)识别方面,期望达到 100% 的识别率,目前尚做不到。所以,在有噪音的环境中,将“tooka”错误的听成为“rooka”是完全可能的。因此,有必要一边对发音的含义进行理解,一边对不合逻辑的单词识别结果进行剔除处理。

(2) **可能存在的一些解释的生成** 所进行的处理是对发音含义的理解。在这种特定的发音场合下,对于“在走廊下读书”这段文字含义的理解以及对其逻辑性的评价,就成了问题的焦点。用计算机去理解自然语言文字的意义时,首先有必要使计算机具有知识,因此,作为第 3 章主题的“知识的表示”,就成了重要问题。在对含义理解结果的表示中,知识的表示也是不可缺少的。“理解”这个问题是很深奥的,从工程学的观点进行的“理解”是肤浅的。可是,对于这个例子来说,作为进行阅读场所的“走廊”,即使不是最适当的场所,但是作为一种可能的场所,也是可以理解的。

(3) **疑问点的记忆** 对不能满足那个解释结果的一些事实,会被记忆到某个地方,以便为今后利用得到的信息,对其进行正确的解释创造条件。这种处理可能被认为是一种特殊的方式,但是毕竟在“理解”时,它可以用已存储(已有)的知识,对理解的对象进行说明(为判定一致性所必须的解释)。当要构造逐步输入信息的理解系统时,这种处理方式,基本上是必不可少的。

(4) **新信息的适当处理** 下面是有关新信息的处理问题。也许这是最关键的问题了。因为根据新的信息推理成何种结果,是完全自由的。在这个例子中,从“以每天 70 页的速度阅读 700 页的书”这个事实,可以推测到阅读的速度很快,英文水平很高(因为那本书是用英文写的),属于粗略地阅读,而且为了阅读该书需要花费 10 天的时间。能够集中时间连续 10 天进行阅读的人,一定是一个有空闲时间的人,等等推论。其中只有一项推论为这次处理带来了重要的新信息。第三章表明,知识表示与推理之间存在

着一定关系,为了得到这里所表示的推理结果,必须准备好与其相适应的知识。

(5) 利用新信息对疑问点进行排除的尝试 怎样利用得到的推理结果呢?在这种情况下,所谓“在走廊阅读”这件事,存在着令人感到勉强的一面,如果觉察到“走廊”一词存在着可疑之处,就可以采取寻找发音与“走廊”相似的推理结果这样一种策略,并且,在这种情况下,还应该能够找到“在10天内阅读”这一情况。这时可以采用在第2章中介绍的“搜索”方法。当然,这时也有可能把“走廊”错误的理解为其他东西,这样就会使问题变得更加复杂。

(6) 意义的一致性判定(修正误解) 一般来说,虽然利用新的信息可以消除含糊不清的东西,但是消除的方法很少是单一确定的,多数情况下存在着两种以上的可能性。而且这时可以根据事先准备好的评价标准(这个标准的表示也是很重要的),从两种以上的可能性中选择出看来是最恰当的一种,然后对过去的解释作出修正。这样做的结果,就使得我们向着正确的解释前进了一步。但是仅仅做到这一步还是不充分的,下面的后处理工作是必须进行的。因为一般情况下是根据过去的解释作出的推理,所以有必要放弃这种推理结果。因为这种推理一旦被取为“真”,就会导致推理结果被取为“假”的情况,所以这种推理被称之为非单调推理。

而且,在实现这个系统时,需要将其变换成第7章论述的人工智能语言的表达形式。关于第4章的机器学习,本例题中虽然没有涉及,但是计算机从例题,或者是从大量的数据中提取出适当的“知识”,作为从事“学习”方面的研究工作是极为重要的,而且是非常有用的。

如果能够掌握本书中所写的内容,是不是可以说就能够构造上述的智能系统了呢?实际上并不那么简单。当然,构造这类系统的可能是有的。然而,作者认为目前要制作出具有如此高的精度,如此强的功能的语言理解系统,实际上是很困难的。最大的障碍是,不清楚应该事先准备些什么样的知识,以及对这些知识应准备到何种程度,因为一面控制来自于新信息的推理,一面迅速地检验出有用的推理结果,是一件非常困难的事情。当需要应用与这些常识有关的知识时,这个问题就会像拦路虎一样阻挡在研究者的面前,为研究者制造困难。人工智能应该研究的课题大多数都还没有解决。

这个问题虽然是从工程学的立场进行处理,但是如果对它作更深入地思考会发现,这里还存在着一些根本性的问题有待解决。到底计算机是否能够真正作到所谓“理解了”这种程度?计算机具有智能这种说法到底是怎样的?下面我们将在回顾历史的同时对这样一些单纯的问题进行解答。

1.2 人工智能的历史

人工智能的观察对象是人类的智能活动。人们注意到,在现今的技术发展阶段,难于实现的人类的智能行为(认识、推理、行动),是通过在计算机上建造计算过程的模型实现的。把重点放在观察和模型化(理解)上的研究,称为科学的人工智能,而把重点放在计算机上进行实施(开发)的研究,则称为工程学的人工智能。

人工智能研究的乐趣(困难),在于在模型化中不受方法论的约束。人脑(在生理学上)是由神经细胞网络构成的。虽然这么说,可是在人工智能的研究方面,还不能用神经网络的机构去说明人脑进行的智能活动的全部内容。虽然希望获得这方面的知识,但是在目前的科学发展水平之下还是不可能完全说明的。例如,利用神经网络的结构,可以对“学会吸氧健身运动的人脑行为”进行某种程度的说明。但是,用神经网络对“在深夜里寻找复印机调色剂的人的行为”进行说明就很困难。但是,却可以推测到,“有必要为第二天早晨发表所需的资料作准备工作”,同时推测到“由于复印机调色剂没有了”,所以会有“在深夜里寻找复印机调色剂”这种行为。这种应用逻辑推理进行说明的方法是比较容易的。最后,在人工智能中有各种各样的模型用来描述人类的智能活动,因此,应根据用途灵活运用。

那么,是否要在所有领域都进行人工智能研究呢?实际上完全不是这样。例如狮子不会学会吸氧健身运动。虽然用神经网络可以说明学会吸氧健身运动时的大脑行为,但是为什么人类能学会吸氧健身运动?若从神经网络的结构上说明这一点仍然是有困难的。有目标的行动称为合理的行动,这时采用逻辑的方法进行说明是比较容易的。但是,另一方面,众所周知,狮子在捕食时,会采取协调行动。当狮子从上风追赶猎物时,潜藏在下风的狮子会伏击正在逃跑的猎物,并且会在适当的时机进行协调出击,以捕获

猎物。看到这种情况的录像后,会使人们感到惊讶。狮子的协调行动,即使可以采用逻辑方法进行说明,但是从直觉上说,人们也很难想像狮子居然也采用了逻辑方法。这样看来,人工智能还仍然处于发展过程中。人工智能的历史,可以说是新的模型生成和被证实的历史,同时也是一种新陈代谢的历史。

那么,什么样的模型才是表示智能活动的良好模型呢?实际上,关于人工智能,还没有对模型的科学的完善性进行过严密地讨论,也没有要求用心理学实验进行验证。为此,各种各样的主意和想法都会受到欢迎。作为模型化的结果,在计算机上实现的应用系统取得了成功,并且进一步考虑了表示该模型意义的有关事项。也许正是因为如此,人工智能的研究历史,与计算机技术的进步历史是同步进行的。不仅要设计模型,而且还要解决在计算机上的实现问题。下面我们一边与本书的内容进行对照,一边简单地回顾人工智能的研究历史。

“人工智能”这一术语,是在1956年的达特莫斯(Dartmouth)会议上,由麦卡锡(McCarthy)首先使用的。麦卡锡因开发出了函数型语言Lisp(详见第7章)而闻名。在达特莫斯会议前后,人工智能的基本理论得到了广泛研究。作为计算机将来的目标,图灵(Turing)提出了图灵测试。

现在对图灵测试进行说明。让人在某一房间内,把计算机配置在另一个房间内。现在把双方都作为“人”来对待,如果在与他们的交往中,不能分辨出他们哪一个是人,哪一个计算机,那么就可以说这台计算机是智能的(intelligent)。这件事还有一段有趣的故事。1965年发表的被称为ELIZA的程序,在社会上产生了强烈地冲击。这时,只要进行一些简单的文章结构方面的操作就会提供出进行自然会话的程序。通过巧妙地借助于对方文章的部分内容,就可以加工出内容广泛的话题。可是,这个系统是否可以称为智能系统?读者这时也许会产生疑问,到底什么是“智能”?如果“智能”指的就是人们看到的那样一些情况,那么ELIZA是有足够的智能的。但是,如果所指的智能是指产生人类合理行动的计算过程,那么ELIZA是不能编排这样的计算过程的。虽然这两方面都是重要的,但是人工智能主要涉及后一种情况。

现在把话题返回到20世纪50年代。当时,受到计算速度和存储容量制约的计算机正处在萌芽时期,国际象棋和西洋跳棋等比赛,尚未成为计算机应用的经常性对象。1950年,奠定了信息

论基础的香农(Shannon),最早写出了关于国际象棋的论文。限于计算机当时的发展水平,无法在走棋方面做到深谋远虑,所以每次观察棋盘上的局势时,都需要研究判断棋局走势的评价函数。最后,当走棋时,还需要作出用数值表达棋盘局势好坏的函数。可是,这种作法,在国际象棋那种复杂的比赛中,实际上起不了多大作用。这一时期,麦卡锡在知识的表示中开始采用谓词逻辑。另外,根据纽厄尔(Newell)和西蒙(Simon)等人的提议^[2]构造了求解疑难问题用的通用问题求解器(general problem solver, GPS)。进入 20 世纪 60 年代后,罗森布拉特(Rosenblatt)提出了一种类似于神经元的元素,它可以实现被称为感知器(perceptron)的模式识别功能。有关感知器和模式识别的问题,将在第 5 章和第 6 章内介绍。

进入 20 世纪 70 年代后,随着计算机的高速化(在过去的 20 年内,计算机的运算速度竟提高了 1 000 倍),国际象棋的研究工作转向了以搜索为中心的计算方面。由于坚持不懈地深入研究走着数,所以在把棋类专家逼入绝境之前,终于取得了进展。这里用到的搜索技术,将在第 2 章内进行介绍。另外,在 20 世纪 80 年代,以丰富的存储容量和人机接口的进步作为背景,对知识的处理研究得到了蓬勃发展。作为描述人类知识的知识表示,以及对其进行高度利用的推理技术,产生出了许多专家系统。第 3 章介绍的知识表示和推理技术,近年来得到了很大发展。在日本,第五代计算机的研究开发中,已着手应用逻辑型语言 Prolog(详见第 7 章)。在国际象棋的程序及其定式和终盘的数据库等中,庞大的数据处理已变得必不可少。1997 年,IBM 公司研制的被称为 Deep Blue 的国际象棋专用系统,终于战胜了人类社会的世界冠军。

现在的人工智能研究工作,区分为两大方向。首先,是以计算机的元件数量接近于人脑的细胞数量为背景的研究,它与生命科学的进展相互呼应,在神经网络、人工生命等领域进行着热烈地讨论。在第 5 章中,将介绍从神经网络和人工生命研究中产生的基本算法;人类梦寐以求的智能机器人,可以作为人工智能具体实现的例子。宠物型机器人在市场上的销售以及机器人足球大赛(RoboCup),已成为人们议论的热门话题。

另一个研究方向,是伴随着计算机网络的进步而形成的。信息网络急剧地扩大了人类社会与计算机的接触面。利用网络促进了大规模知识库的综合和再利用。在机器学习成果的基础上,对

大量数据进行处理的数据开发工作,也随之活跃起来。通过对人类基因和组合存储器营业额信息的分析,人们试图寻找出隐藏在庞大的数据中的法则。在第4章中,将说明这种机器学习的基本技巧。对用户提供支援工具的开发工作也在进行之中。在网络内的三维假想空间中,销售新车的公司营销人员,正面带笑容地开始进行商务谈判。

这样,在人工智能方面,迄今已经提出了许多种模型方案,形形色色的应用系统被开发出来。这种开放性的研究特色,有时也会遭到来自计算机科学主流学派的批判,但是那些朝气蓬勃的研究人员和工程技术人员,仍然在不断地得到用户的支持与关注。他们肩负着向计算机科学的梦幻世界继续开拓的使命。学习完本书以后,对于那些希望进行更深入学习的读者,建议参考本书后面列出的参考文献^[31]。

第 2 章

基于搜索的问题求解

人工智能要解决的问题,大部分不具备明确的解题步骤。这类问题可以采用搜索方法解决,但是在解决过程中,会伴随着产生试行错误。本章将对采用状态空间图的问题表示方法、采取纵向搜索和横向搜索的系统的搜索方法、运用启发式(已发现的知识)的搜索方法、表示问题分解的与(AND)/或(OR)图的搜索,以及在计算机国际象棋等方面用到的博弈树搜索,进行分析和说明。

2.1 搜索与人工智能的关系

2.1.1 八数码魔方

计算机具有的计算能力远远超过人类,但是它的计算顺序必须由程序明确地表示出来。然而在日常生活中,多数问题不具备明确的计算(或解题)步骤。

例如在图 2.1 中所示的数字魔方中,它由八块方砖构成,称为八数码魔方。现在将排列成无序状态的方砖,进行上下左右方向的挪动,使其变成按数字顺序排列的模式¹⁾。这种游戏初看似乎很简

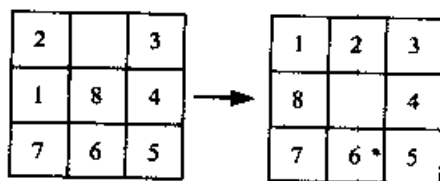


图 2.1 八数码魔方

单,但是由于没有明确的解题步骤可供选择,所以为了能正确地摆放方砖,需要花费相当长的时间。再者,作为对阵的棋手,在进行象棋和国际象棋比赛的情况下,还必须根据对手的棋子走法,不断

1) 方砖如果能按顺序排列,则可以显现出图案来。另外,方砖的数目为 4×4 和 5×5 时的魔方,分别被称为 15 数码魔方和 24 数码魔方。方砖的数目无论怎样增加,都是可以的,但是相应问题的求解也会变得更加困难。

地改变自己的棋子走法,因此会使解法变得更加复杂。

当然,因为求解方法是不明朗的,所以只有当游戏和比赛精彩而有趣,而且确信存在着取胜方法时,才能做到引人入胜,否则谁也不会对其发生兴趣。在人工智能处理的问题中,这种求解步骤不明朗的情况是很多的,解决这类问题的方法之一,就是采用搜索法。

2.1.2 状态空间表示

为了用计算机解决问题,首先有必要用程序去表示那些可以进行表示的问题。在解题方法的步骤明朗时,用流程图(流图)进行表示,但是在解题方法的步骤不明朗时,可以采用状态空间表示作为问题的表示方法。在这种情况下,我们以机器人从容易迷失

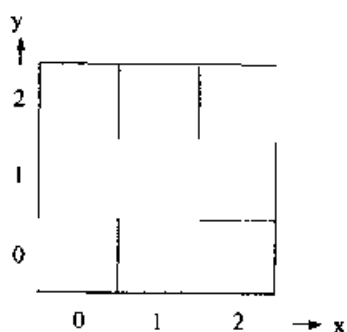


图 2.2 迷 宫

方向的格状路段迷宫(也可称为盘陀路)入口(坐标为 $(0,0)$),到出口(坐标为 $(2,2)$)的运行过程为例,对状态空间表示方法加以说明。机器人最初位于入口处,它可以向着相邻的可能进行移动的任何坐标移动。这样在能够到达出口之前,就产生了路径选择问题。

在状态空间表示法中,要求解的问题是状态和算符集合的形式表示的。

在上述问题中,对应于机器人的位置,可以定义出 9 种状态。例如,机器人在坐标 $(0,1)$ 的位置上可以用状态 $(0,1)$ 表示。所谓算符是一种表示状态与状态关系的符号,在本问题中,相应于机器人的移动方向上(U)下(D)左(L)右(R),可以定义四个算符。例如,对应于状态 $(0,1)$,可以应用算符 U, D, R。其应用结果是分别可以变化到状态 $(0,2)$, $(0,0)$, $(1,1)$ 。这种状态与算符的关系可以用图 2.3 所示的状态空间图表示。

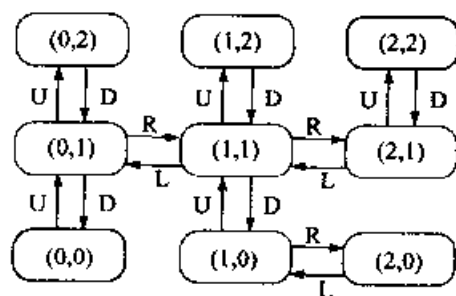


图 2.3 迷宫的状态空间表示

不论是什么问题,都具有求解前的状态和求解后的状态,前者称为**初始状态**,后者称为**目标状态**。在迷宫问题中,机器人进入入口的状态 $(0,0)$ 是初始状态,进入出口的状态 $(2,2)$ 是目标状态。而且所谓的问题求解,就是依次应用可以应用的算符,将初始状态变化到目标状态。在图 2.3 所示的例子中,如果对初始状态依次应用算符 U, R, R, U, 则可以变化到目标状态。

2.1.3 与图有关的术语

图 2.3 所示的状态空间,可以用图来表示,问题的求解也可以看作是在这种图上的搜索。作为以后讨论中的准备工作,首先就有关状态空间图的术语进行归纳整理。

状态空间图是由节点的集合和分枝的集合构成的。节点数目有限的图称为**有限节点图**。一个分枝连接两个节点。在分枝中,有具有方向的分枝(有向分枝)和不具有方向的分枝(无向分枝)。无向分枝与具有各自方向的两个有向分枝的意义相同。当存在从节点 n_i 开始指向 n_j 的分枝时,则称 n_i 为 n_j 的**双亲节点**, n_j 为 n_i 的**子节点**。求解节点 n_i 的所有子状态,称为将节点 n_i **扩展**。当节点的系列为 n_1, n_2, \dots, n_m , 且存在着从各个节点 n_i 开始指向 n_{i+1} 的分枝时($1 \leq i \leq m+1$), 则称之为从 n_1 到 n_m 的**路径**。路径中不包含两个以上的相同分枝,两端的节点如果是相同的,则称这种路径为**闭路**(参考图 2.4(a))。

对于所有不同的两个节点,不考虑分枝的方向,把它们连结起来的路径构成的图称为**连结图**。不构成闭路的连结图称之为**树**。

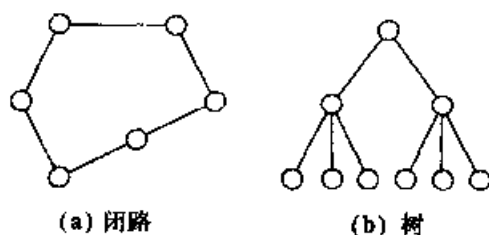


图 2.4 闭路和树举例

在用图来表示状态空间时,可以相应的用节点表示状态,用分枝表示算符。这时与初始状态对应的节点称为**初始节点**,与目标状态对应的节点称为**目标节点**。因此,所谓的问题求解,就是求出从初始节点到目标节点的路径(即解)。对于图 2.3 来说,从初始

节点 $(0,0)$,到目标节点 $(2,2)$ 的序列 $(0,0),(0,1),(1,1),(2,1),(2,2)$,就变成了问题的解。

2.2 逐个搜索

解决问题,意味着找到从状态空间图的初始节点到目标节点的路径。因此,实际上是应当说明为找到上述路径所需要的算法。现在首先从最简单的问题开始讨论,然后再对其进行修正,逐步过渡到高级的问题。

2.2.1 随机搜索

用来寻找最单纯的路径的算法,是下面表示出来的随机搜索方法¹⁾。

算法1 随机搜索

- 第1步:设初始节点为 n ;
- 第2步:如果节点 n 为目标节点,则求解以成功告终;
- 第3步:扩展节点 n ,得到子节点的集合;
- 第4步:从子节点中选择一个节点,并且设为 n' ;
- 第5步:设 n' 为新的 n 并进入第2步。

这种方法是一种从初始节点开始,然后接连不断地扩展节点,直到达到目标节点的方法。如果到达了目标节点,则扩展节点的顺序就构成了解²⁾。

对于这种算法的第4步来说,在其后进行扩展时,并没有特别的规律去指导如何选择子状态。例如,退出循环的决策是无关紧要的。但是应用这种方法时,在到达目标状态以前,也许会遇到不走运的情况,一般来说,不能够保证一定能达到正确的目标节点。总之,存在着算法不能终止的情况。例如在图2.3中的极端情况下,也许会发生算法在节点 $(0,0)$ 与 $(0,1)$ 之间反复循环的情况。

2.2.2 CLOSED表的引入

为避免发生上述情况,可以采取一些措施,使得同一个节点不

1) 利用这种算法实现的随机搜索方法,又称为随机游动方法。

2) 当然,在这个解中包含了闭路的情况。

会被两次扩展。为此,可以把扩展过一次的节点,记录到 **CLOSED** 表中,从而使其不再成为以后扩展时的候选对象。于是新的算法就变成了下列形式。

算法 2 **CLOSED** 表的引入

第 1 步:设初始节点为 n ;

第 2 步:如果节点 n 为目标节点,则求解以成功告终;

第 3 步:扩展节点 n ,得到子节点的集合。将 n 加进 **CLOSED** 表中;

第 4 步:从子节点的集合中选择一个未包含在 **CLOSED** 表中的节点,并设其为 n' 。如果不存在这样的节点,则求解以失败告终;

第 5 步:设 n' 为新的 n 并进入第 2 步。

在算法 2 中,同一状态不可能两次扩展,于是可以导出下列命题。

命题 2.1 如果搜索图是有限的,则算法 2 必定会停止。

〈证明〉 如果搜索图是有限的,则节点数也是有限的。因为算法的扩展数不超过节点数,所以会停止下来。

根据命题 2.1,在有限搜索图的情况下,虽然停止性可以得到保证,但是却有可能发生得不到解的问题。例如在图 2.3 中,设按照节点 $(0,0)$, $(0,1)$, $(0,2)$ 的顺序进行扩展。这时作为 $(0,2)$ 的子状态,虽然能够得到状态 $(0,1)$,但是因为这个状态已经包含在 **CLOSED** 表内,所以在第 4 步中会以失败而告终。即走到了尽头而不能退出,这是这种算法的缺点。

2.2.3 OPEN 表的引入

鉴于上述情况,引进了新的 **OPEN** 表,以便能从走入尽头的情況中退出来。**OPEN** 表用来存放以后能够扩展的候选状态。新的算法这时变成为下列形式的算法。

算法 3 **OPEN** 表的引入

第 1 步:将初始节点存放到 **OPEN** 表中;

第 2 步:从 **OPEN** 表中取出节点,并且设其为 n 。如果 **OPEN** 表是空的,则求解以失败告终。如果节点 n 是目标节点,则求解以成功告终;

第 3 步:扩展节点 n ,得到子节点的集合。将 n 加到 **CLOSED** 表中;

第4步:对于子节点集合中不包含在CLOSED表中的节点,配置指向节点 n 的指针;

第5步:返回第2步。

在这种算法中,扩展的节点顺序,并不构成原来求解的路径。因此,是通过指针(从子节点指向双亲节点的转移指令)来表示扩展的双亲节点与子节点的关系。当到达目标节点时,通过追踪指针就可以得到求解路径。这个算法如下面表示的那样,其搜索图是有限的,如果解存在,则必定能找到解的这种所谓安全性会得到保证。

定理 2.1 对于解存在的有限搜索图,算法3必然能找到解。

〈证明〉 下面两种情况中的任何一种都是不能找到解的情况。(a)陷入无限循环之中;(b)在第2步中,OPEN表变空,从而以失败告终。因为根据命题2.1不可能发生情况(a),所以假定是情况(b)。由于必定存在着解,所以可以设其节点序列为 n_1, n_2, \dots, n_m 。这里 n_1 是初始节点, n_m 是目标节点。在这些节点中,设未包含在CLOSED表中且下脚标最大的节点为 n_i 。于是 n_i 必然包含在CLOSED表内,因为 n_m 不包含在CLOSED表内,所以上述 n_i 必然存在。这时 n_{i-1} 包含在CLOSED表内。这样一来就能对 n_{i-1} 进行扩展,作为其子状态之一的 n_i 理应包含在OPEN表内。因为 n_i 未包含在CLOSED表内,而且尚未被扩展,所以OPEN表不会是空的。这与(b)中的最终结果相矛盾。

2.2.4 纵向搜索

在以前的算法中,对于从OPEN表取出节点的方式,没有作过特别的限制,但是在这里,我们要假设OPEN表是“后人先出”的存储栈。这时,图2.3所示状态空间图中的搜索过程变成为图2.5那样的情况。这里在对节点进行扩展时,假定将子节点放入OPEN表的顺序为图2.3中的上(U)、右(R)、下(D)、左(L)的状况。在图2.5的情况下,被扩展节点的顺序,和那时的OPEN表、CLOSED表的值,均标记于图上。这里因为OPEN表是右侧为出入口的存储栈,所以为了便于进行展开,应注意对表的后面节点的选择。

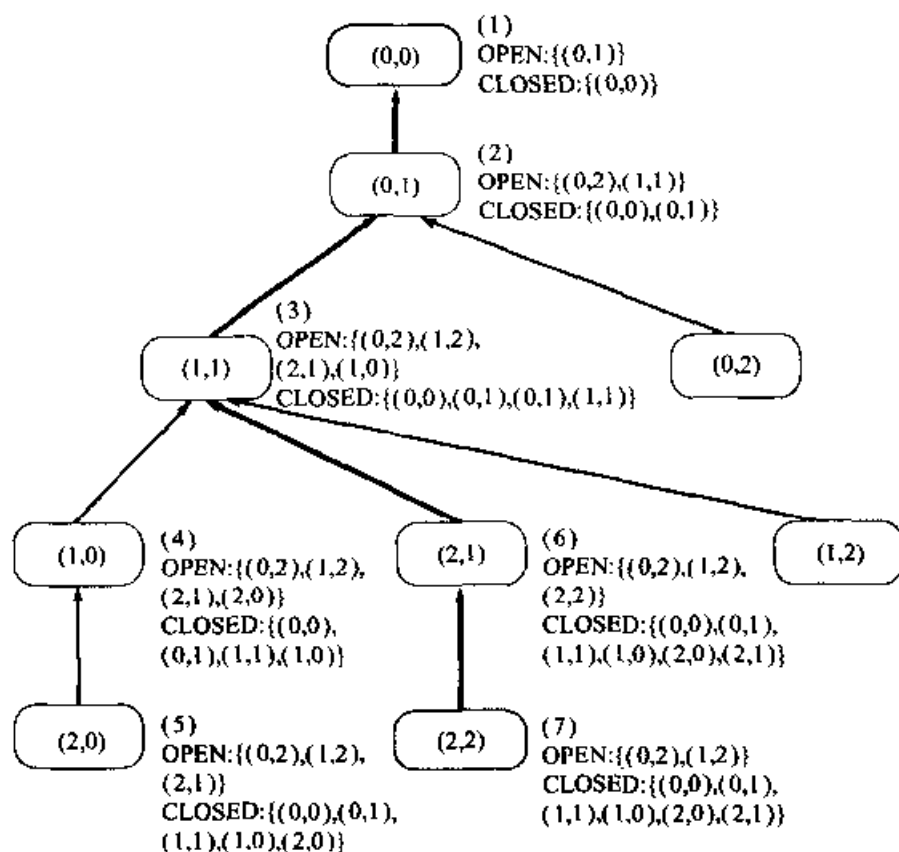


图 2.5 纵向搜索过程

这种算法称为纵向搜索(或深度优先搜索)。这种算法的特点是,要一个一个新得到的子节点进行扩展,直到不可能再进行扩展为止。因为纵向搜索保持了算法 3 的性质,所以在节点图为有限的情况中,停止性和完全性将得到保证。可是在图 2.6 那样的图为无限节点的情况下,将会出现所谓算法不能停止的问题。

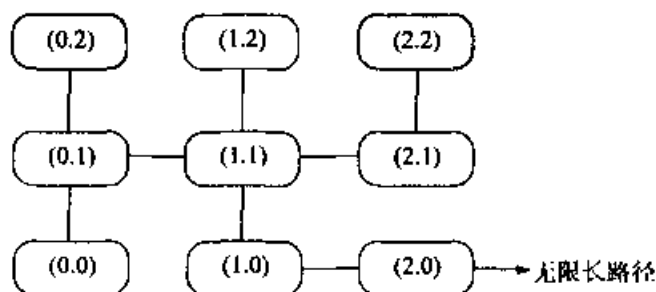


图 2.6 无限长路径图

2.2.5 横向搜索

下面我们考虑假设 OPEN 表以“先入先出”的原则排队时的

情况。这时图 2.3 中状态空间图的搜索过程变成为图 2.7 所示的搜索过程。

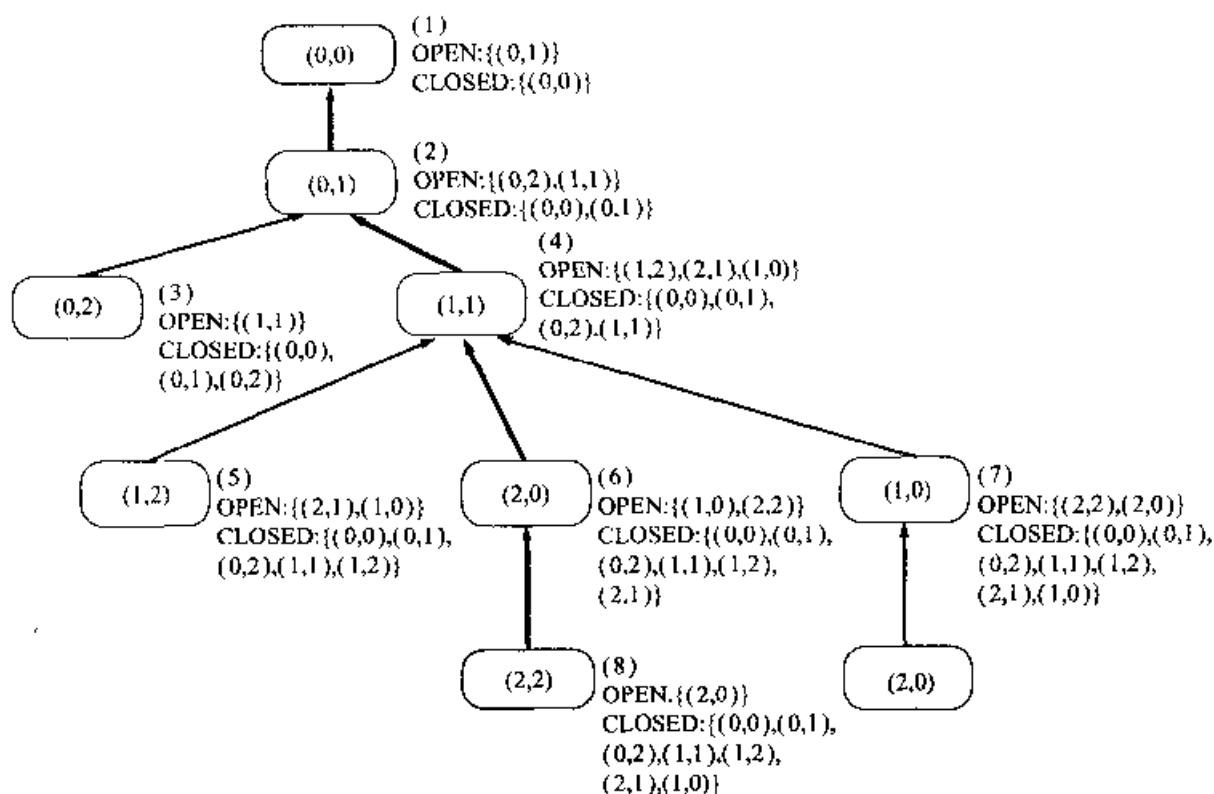


图 2.7 横向搜索过程

这种算法称为**横向搜索**(或**广度优先搜索**)。这种算法的特点是,对展开得到的所有子节点都进行扩展后,再对新的子节点进行展开。横向搜索与纵向搜索不同,例如即使设节点图为无限的,如果解存在,那么这种算法就能够求出这个解。在后面介绍的定理 2.2 中,上述结果作为一种特殊情况可以得到证明。这里用图

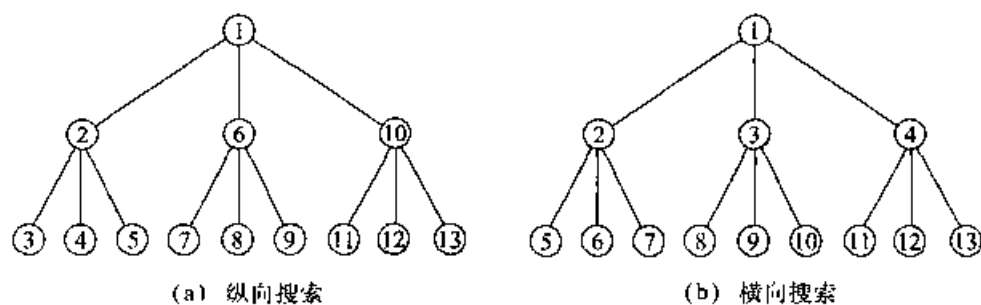


图 2.8 纵向搜索与横向搜索

2.8 中表示的节点树,清楚地揭示了纵向搜索与横向搜索的区别,在图中显示了它们不同的扩展顺序。

2.2.6 均一代价搜索

从此开始,我们来处理必须考虑解的代价的问题。例如在迷宫问题中,会遇到希望以尽可能短的距离到达出口的情况。解的代价是以导出解的算符的代价之和来表示的,这种算符的代价,可以用图 2.9 中表示的分枝的代价表示。用 $C(n_i, n_j)$ 表示从节点 n_i 到 n_j 的分枝的代价。另外,设分枝的代价为正值。例如,从 F 到 G 的分枝代价为 1,且用 $C(F, G) = 1$ 表示。因此,当设初始节点为 F,目标节点为 B 时, $F-G-E-C-A-B$ 就是一个解,它的代价为 $1+1+1+1+2=6$ 。另外, $F-G-E-A-B$ 也是一个解,它的代价为 $1+1+4+2=8$ 。如果不存在比某个解 s 的代价更小的解,则 s 就被称为**最优解**。在这个例子中, $F-G-E-C-A-B$ 就成了最优解。

这里让我们来说明求最优解的**均一代价搜索**算法。搜索的方针是起始于初始节点,并从路径代价更小的节点进行扩展。现在我们用 $g^*(n)$ 表示从节点 n 的初始节点开始的最优路径。另外,在搜索过程中采用的估计值用 $g(n)$ 表示。此外,设将节点及其(估计)代价记录到 OPEN 表中,并且按照代价的递增次序进行排序。

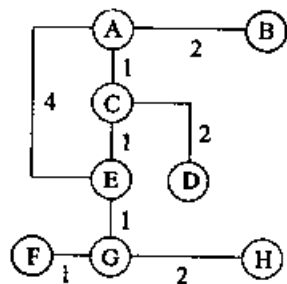


图 2.9 标出代价的节点图

算法 4 均一代价搜索

第 1 步:将初始节点 n_i 及其代价 $g(n_i) = 0$ 放入 OPEN 表中;

第 2 步:把 OPEN 表中最前面的节点取出来,并且设为 n 。如果 OPEN 表是空的,则求解以失败告终。如果节点 n 是目标节点,则求解以成功告终;

第 3 步:扩展节点 n ,得到子节点的集合。将 n 放入 CLOSED 表;

第 4 步:对子节点的集合中不包含在 CLOSED 表中的节点 n' ,配置指向 n 的指针。计算 $g(n') = g(n) + c(n, n')$,并且将其放入 OPEN 表。但是 n' 已经放入 OPEN 表,当新的 $g(n')$ 比估的值小时,则应予以更新,指针也应予以更换。对于 OPEN 表应依照

代价的递增次序进行分类；

第5步：返回到第2步。

根据上述算法，可以将图2.9中表示的节点图的搜索情况表示成图2.10。OPEN表的元素是用节点名及其代价对来表示的。在对节点E进行扩展时，节点A被放入OPEN表，对C进行扩展时， $g(A)$ 的值和指针的更换，都是需要注意的事项。图2.10中，归的指针用虚线表示。

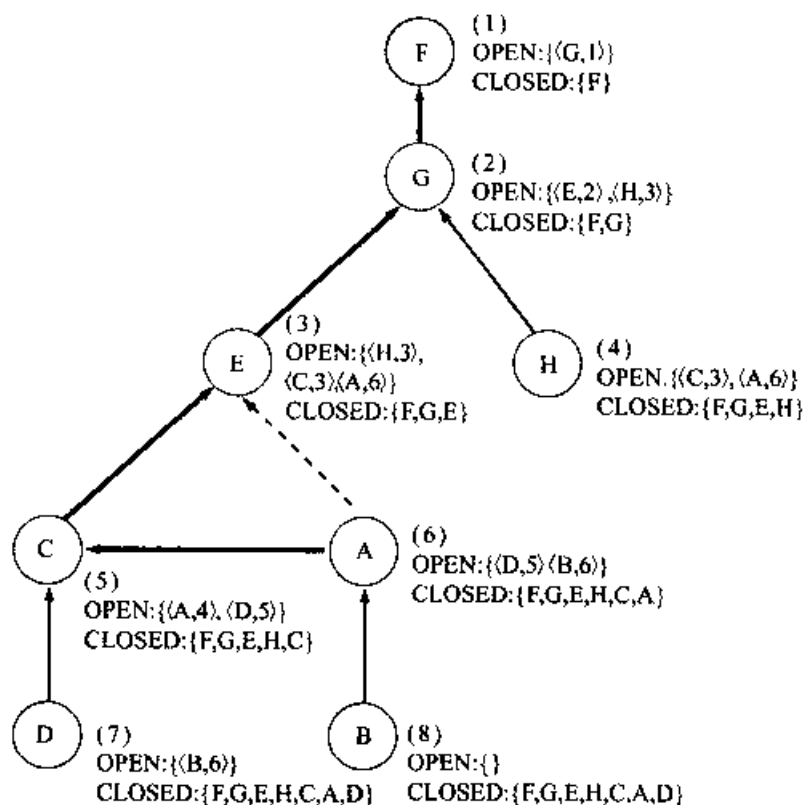


图 2.10 均一代价的搜索过程

在均一代价搜索的情况下，第4步中放入CLOSED表中的节点，没有必要再放入OPEN表中。这可以从下面的定理导出。

定理 2.2 均一代价搜索对节点 n 进行扩展时，通往节点 n 的最优路径便已经被找到。即 $g(n) = g^*(n)$ 。

〈证明〉 如能证明OPEN表中最前面节点的 $g(n)$ 为非衰减函数，则定理即得到证明。在均一代价搜索的情况下，由第2步到第5步的循环过程中，OPEN表中最前面的节点 n 将被除去，而其子节点的集合将被加进来，所以根据需要，OPEN表内节点的代价会产生变化。对于节点 n 的子节点 n' ，等式 $g(n') = g(n) + c(n, n')$ 成立，因为

$c(n, n') > 0$, 所以 $g(n') > g(n)$ 。于是在 OPEN 表中被添加的节点, 或者说代价发生了变化的节点的代价, 其总和是非衰减的。因此, 将来不可能找到比 OPEN 表中最前面的节点 n 的代价 $g(n)$ 更小的代价, 所以 $g(n)$ 是最小代价。

另外, 即使是对于无限图, 因为也能够保证算法会终止(证明略), 所以只要均一代价搜索存在最优解, 则必定找到这个解的所谓可纳性会得到保证。这里如果假定图中分枝的代价全为 1, 则这种算法与横向搜索是一致的。

2.3 应用智能的搜索

2.3.1 启发式搜索

以前谈到的算法, 只利用标有图的形状和代价的状态空间图直接表示出来的信息。因而在状态空间变大时, 与之相应的搜索所需的时间和存储量都会相应地增大。在人工智能处理的问题中, 同样存在着状态空间非常大的问题。例如在图 2.1 中表示的 8 数码魔方中, 存在有 $9!/2 = 181440$ 种不同的状态。在 15 数码魔方中, 这个数目会变得更大, 这时它可以达到 $16!/2$ 种不同的状态, 如果采用一个不漏地进行试验的搜索方法获得求解结果, 将会是一件很困难的事情。

因此, 尽可能地减小搜索范围的方法变得越来越重要。为此, 出现了应用启发式(应用已有知识)的方法, 这种搜索方法被称为启发式搜索。例如在迷宫问题中, 如果知道目标状态的位置, 那么就应该考虑采取尽可能靠近该位置的搜索策略。针对该问题的一种方法, 是当设定目标节点的坐标为 (x_k, y_k) 时, 利用从节点 $n = (x, y)$ 到目标节点的曼哈顿距离 $h(n) = |x_k - x| + |y_k - y|$, 对能使这个距离值变小的那种节点进行扩展。

不过, 这种启发式的方法, 虽然在多数情况下是有用的, 但是未必都是正确的。例如在图 2.9 所示的那种迷宫中, 对节点 G 进行扩展时, 存在着两个可以进行扩展的候选节点, 即 $E = (1, 1)$ 和 $H = (3, 0)$ 。这时, $h(E) = |3 - 1| + |3 - 1| = 4$, $h(H) = |3 - 3| + |3 - 0| = 3$, 虽然可以说 H 一方离目标节点近, 但这却不是正确的选择。

2.3.2 登山法和最佳优先搜索

登山法是采用启发式的一种最简单算法。现在我们来介绍这种方法。

算法5 登山法

第1步: 设初始节点为 n ;

第2步: 如果节点 n 为目标节点, 则求解以成功告终;

第3步: 扩展节点 n , 得到子节点的集合;

第4步: 从子节点的集合中, 选取 $h(n')$ 为最小的节点 n' 。如果 $h(n) < h(n')$, 则求解以失败告终;

第5步: 设 n' 为新的 n , 返回第2步。

在这个算法中, 如果把 $h(n)$ 看作是到达山顶(目标节点)的距离, 则因为 $h(n)$ 的值是向着减小的方向进行搜索, 所以看起来恰似进行登山时的情况。不过当这样的节点不再存在时, 就会以失败而告终。即当登山途中遇到小的山峰时, 就会产生不能由此继续完成原来进程的问题。在图 2.9 中的节点 G 以后, 若选择 H, 则此后就会发生走投无路的情况。这种情况可以说与前面介绍过的随机搜索中产生的问题是相同的。因此, 人们试探地构成了加进 OPEN 表和 CLOSED 表的算法。这里对于前面的算法 4, 可以只根据 $g(n)$ 的替代量 $h(n)$ 的值, 对 OPEN 表进行分类。这时的搜索过程如图 2.11 所示。

这种算法称为最佳优先搜索。最佳优先搜索不必担心以前的代价, 只需选择能使将来的代价预测值变成最小的路径。这种算法可以保证能够找到解, 而且它可以在许多场合中高效地运行。如果把这种搜索与均匀代价搜索进行比较, 则可知节点 C 和 D 此时不会被扩展, 而且显然只通过 6 次扩展就可以求出解来。不过由这个结果也可以发现, 最佳优先搜索不能保证获得最优解。

2.3.3 A* 算法

为什么用最佳优先搜索算法不能得到最优解呢? 当考虑这个问题时, 我们发现, 这是因为忽略了始于初始节点的代价造成的。因此, 节点 n 的评价值可以定义如下:

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

这里, $g(n)$ 是从初始节点到节点 n 的路径代价的估计值, $h(n)$ 是从节点 n 到目标节点的路径代价的估计值。设 $f(n)$ 为评价函数, 则能满足 $h(n)$ 的估计值是实际的最优值 $h^*(n)$ 的下界函数这一条

件¹⁾的搜索算法,称为 A* 算法。

算法 6 A* 算法

第 1 步:将初始节点 n_i 和它的代价 $f(n_i) = g(n_i)$ 放入 OPEN 表中;

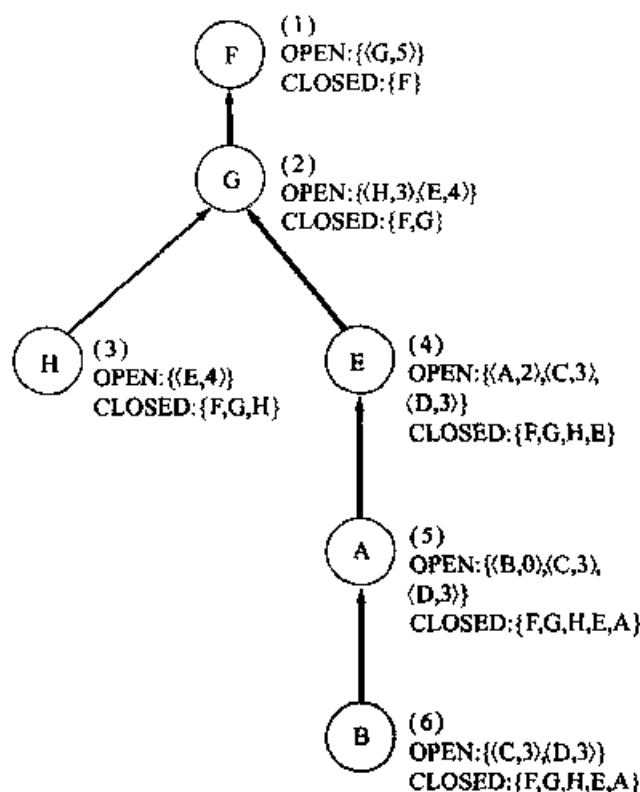


图 2.11 最佳优先搜索过程

第 2 步:把 OPEN 表中最前面的节点取出来,并设其为 n 。如果 OPEN 表是空的,则求解以失败告终;

第 3 步:扩展节点 n ,得到子节点的集合,把 n 放入 CLOSED 表;

第 4 步:对于子节点的集合中不包含在 CLOSED 表中的节点 n' ,配置指向 n 的指针。计算 $g(n') = g(n) + c(n, n')$,并且放入 OPEN 表中。但是在 n' 已经放入 OPEN 表的情况下,当新的 $g(n')$ 比估的值小时,则应予以更新,指针也应予以更换。在 n' 已经放入 CLOSED 表中的情况下,当新的 $g(n')$ 比估的值小时,则应从 CLOSED 表中取出,放入 OPEN 表,并且更换指针。对于 OPEN 表,按照评价值递增的顺序进行分类;

1) 迷宫的曼哈顿距离,就是满足这一条件的估计值。

第5步:返回到第2步。

搜索过程如图 2.12 所示。与均一代价搜索比较,这里不对 D 进行扩展。显然,这时只通过 7 次扩展就可以得到解答。但是在 A* 算法中,正如第 4 步表示的那样,会发生放入 CLOSED 表中的节点被再次放入 OPEN 表的情况。具体的例子请读者参考本章的练习题 3。

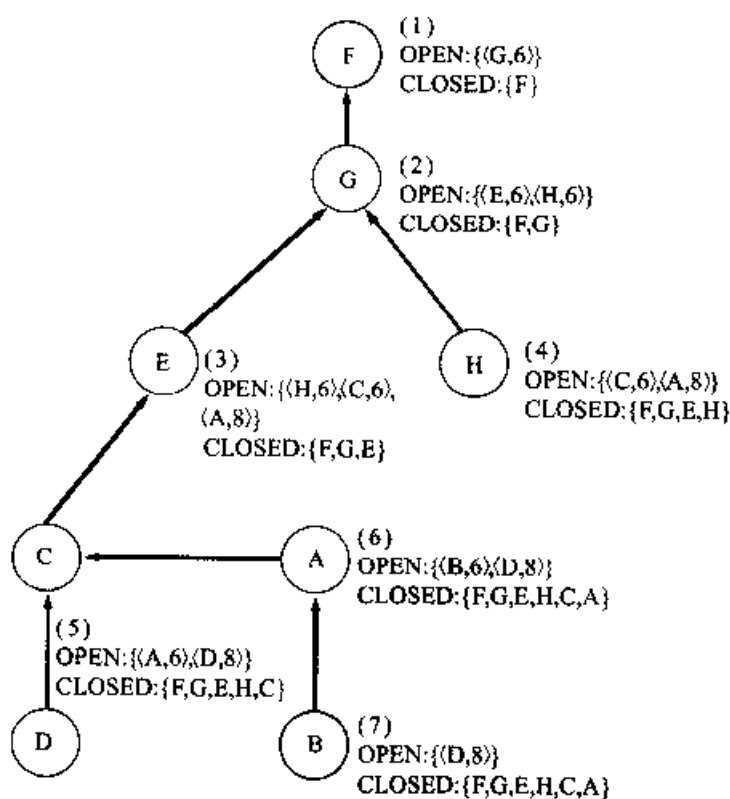


图 2.12 A* 算法的过程

在 A* 算法中, $h(n) \leq h^*(n)$ 是能够找到最优解的必要条件。不能满足这个条件的算法称为 A 算法, 它不能保证得到最优解。图 2.13 是它的一个例子。另一方面, 如果 A* 在状态空间图中的

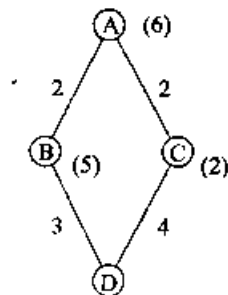


图 2.13 $h(n) > h^*(n)$ 的图

解存在, 则能保证最终可以求得最优解。

另外, 设存在着两个 A* 算法 A_1 和 A_2 , 并设它们的启发函数分别为 h_1 和 h_2 , 如果对于所有的节点 n , 满足 $h_1(n) > h_2(n)$, 则可以证明由 A_2 扩展的节点数比由 A_1 扩展的节点数多。

这表明启发函数更加接近于真值,于是搜索效率便越高。即如果 $h(n)=h^*(n)$,则这时的解可以说就表示一个预先明确的问题了。此外,在 $h(n)=0$ 的情况下,它属于一种完全没有提供有关解的信息的情况,这时它变为与均一代价搜索等价的情况。

2.3.4 约束的利用

通过利用启发式函数,可以使 A* 算法的搜索范围变小,搜索效率提高。在这里,我们可以把利用约束条件,作为进一步减小搜索范围的手段。例如,从 1 到 4 的整数值中,取出三个值作为变量 x, y, z 的值,试考虑如何解这些值构成的组合问题。对于具体由哪些值的组合构成解的问题,这里先不涉及,但是至少对于变量作出了下列约束:

$$x+y \geq 7 \quad (2.1)$$

$$x+z \geq 5 \quad (2.2)$$

解决这类问题的最简单方法,是使 x, y, z 的值的组合,按照 $(1,1,1), (1,1,2), (1,1,3)$ 这样一种方式,顺序地进行改变,并且检验它们是不是满足约束条件的解。这时,设由各个变量可能得到的值的集合用 L_x, L_y, L_z 表示,则因为 $L_x=L_y=L_z=\{1,2,3,4\}$,所以需要进行检验的最大组合数为 $4^3=64$ 种。

可是,如果利用由两个不等式表示的变量间的约束条件,则组合的范围会被缩小。例如,根据(2.1)式,显然 x 与 y 同时为 1 和 2 的情况是不可能发生的。即 $L_x=L_y=\{3,4\}$ 。此外,根据(2.2)式,显然 $L_z=\{2,3,4\}$ 。因此,在这种情况下,通过对解进行检验可知,变量的组合数减少到 $2 \times 2 \times 3=12$ 组。由于利用了附加到这个问题上的约束条件,使得搜索范围缩小,从而有可能提高 A* 算法的效率。

2.4 对问题进行分割后进行搜索

2.4.1 与/或(AND/OR)图表示

此前的搜索方法,都是使初始状态(初始节点)通过接连不断地变化,逐渐追寻,最后达到目标状态(目标节点)。与此相应,也可以根据问题的具体情况,考虑采用先将其分解成几个问题,然后分别进行求解的方法。这种问题的求解方法,可以作为 AND/OR

图搜索而将其规格化。作为 AND/OR 图搜索的具体例子,我们来说明文章的构成问题。这里是根据设定的文法规则来构成文章的。例如下列一些文法。

R1:〈名词句〉→〈冠词〉〈名词〉

R2:〈谓语〉→〈动词〉〈名词句〉

R3:〈主语〉→〈名词句〉

R4:〈主语〉→〈代名词〉

R5:〈文〉→〈主语〉〈谓语〉

R6:〈名词〉→man

R7:〈名词〉→mouse

R8:〈代名词〉→she

R9:〈动词〉→caught

R10:〈冠词〉→the

这里包含在〈 〉中的符号称为非终端符号,不包含在〈 〉中的符号称为终端符号。由〈文〉开始,并且排出仅由终端符号构成的符号列,就是给出的问题。这里,文法规则 R5 表示〈文〉是由〈主语〉和〈谓语〉的符号组成的,为了求出满足〈主语〉和〈谓语〉各自的符号列,可以用 AND 分割分解为两部分问题。利用这种方法可以保证,当被分割的所有部分的问题被解决时,原来的问题也得到解决。

另外,R3 和 R4 表示满足〈主语〉的符号列,是〈名词句〉或〈代名词〉中的任一种符号列,可以用 OR 分割予以表示。利用这种方法可以保证,当被分割的任意一部分问题被解决时,原来的问题也被解决。

上述根据文法规则构成的文章生成问题,可以用 AND/OR 图表示成图 2.14 所示形式。在这里 AND/OR 分割可以分别用 AND/OR 分支来表示,为了对它们进行区别,在 AND 分支中添加了圆弧。由 AND 分支得到的子节点称为 AND 接点,由 OR 分支得到的子节点称为 OR 节点。另外,最初给出状态的节点,称为初始节点。AND/OR 图的端点变成为部分解作为目标节点,也可能是相反情况,不作为目标节点。在文章生成方面,初始节点为非终端符号〈文〉,终端符号变为目标节点。这里值得注意的是,在 AND/OR 图搜索中,在刚一发现目标节点的情况下,是不能说已经找到了解的。

AND/OR 图搜索的解称为解图,它是原来的 AND/OR 图的

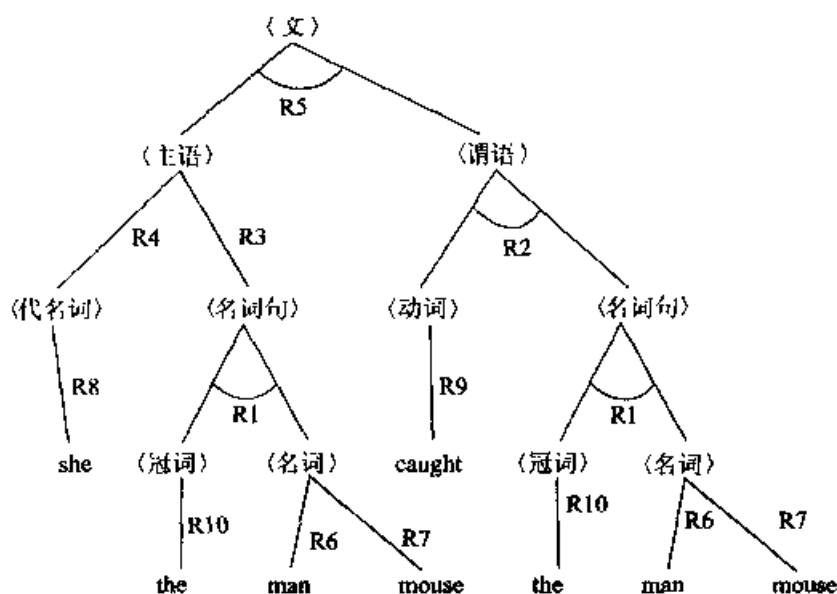


图 2.14 表示文法规则的 AND/OR 图

一部分,并且具有下列性质。

- (a) 包含初始节点;
- (b) 在某节点具有 AND 分支的情况下,它将包含其所有子节点;
- (c) 在某节点具有 OR 分支的情况下,它将包含其所含子节点中的一个子节点;
- (d) 所有的终端节点都是目标节点。

图 2.15 表示了一种文章生成问题的解图,一般来说,对应于一个 AND/OR 图,存在着多种解图。AND/OR 图搜索,是用来寻找由问题定义的 AND/OR 图中的解图的。

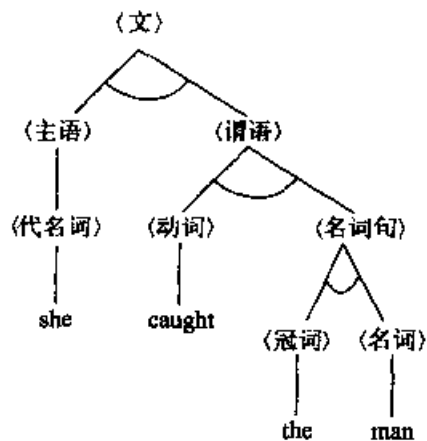


图 2.15 解图

2.4.2 与/或(AND/OR)图搜索

AND/OR 图搜索是从仅由初始节点构成的未解的图开始,到获得解图的过程中,反复进行的扩展与评估操作。因为未解的图在搜索过程中存在着多种情况,所以采用了与以前的搜索算法相同的 OPEN 表,对这些情况进行管理。于是 AND/OR 图搜索算法变成了下列形式。

算法 7 AND/OR 图搜索

第 1 步:把仅由初始节点构成的未解图放进 OPEN 表;

第 2 步:从 OPEN 表中取出一个节点,设为 g 。如果 OPEN 表是空的,则求解以失败告终,如果 g 是解图,则求解以成功告终;

第 3 步:扩展 g ,得到未解图的集合。对这些未解图进行评估,并将无解图以外的子节点放入 OPEN 表;

第 4 步:返回第 2 步。

现在我们对第 3 步中的 AND/OR 图扩展和评估进行说明。AND/OR 图的扩展,是在未解图的端点,将后面谈到的未加标记的节点进行扩展。其子节点如果是 AND 节点,则对所有的子节点施加作用。如果其子节点是 OR 节点,则只对各节点中的一个节点施加作用,并且仅由施加作用的子节点数生成未解图。

另外,未解图的评估可以按以下方式进行。对未解图的各端点进行审查,如果是目标节点,则加上 S 标记。此外,如果是不能作进一步扩展的端点,则加上 N 标记。对于端点以外的节点,当其节点的子节点为 AND 节点的情况,且其所有子节点为 S 时,则应加上 S 标记。但是即使有一个子节点为 N 时,则应加上 N 标记。另外,当子节点为 OR 节点的情况时,如果它是 S,则应加上 S 标记,如果是 N,则应加上 N 标记。如果初始节点变成为 S,则为解图,如果变成为 N,则为无解图。另外,如果未加任何标记,则仍然是原来的未解图。

取图 2.14 所示的 AND/OR 图作为例子,其 AND/OR 图的搜索过程表示在图 2.16 上。在 AND/OR 图的搜索方面,关于从 OPEN 表中提取未解图的方式,曾经考虑过各种方案,但这里只表示了采用纵向搜索的例子。图中的 S 表示标记。图中作了一些省略,在扩展(b)时,不仅在(c)中端点变成了〈代名词〉,而且还产生了端点变成〈名词句〉的情况。另外在对(h)进行扩展时,同样也得到了两种图,这是应当注意的问题。

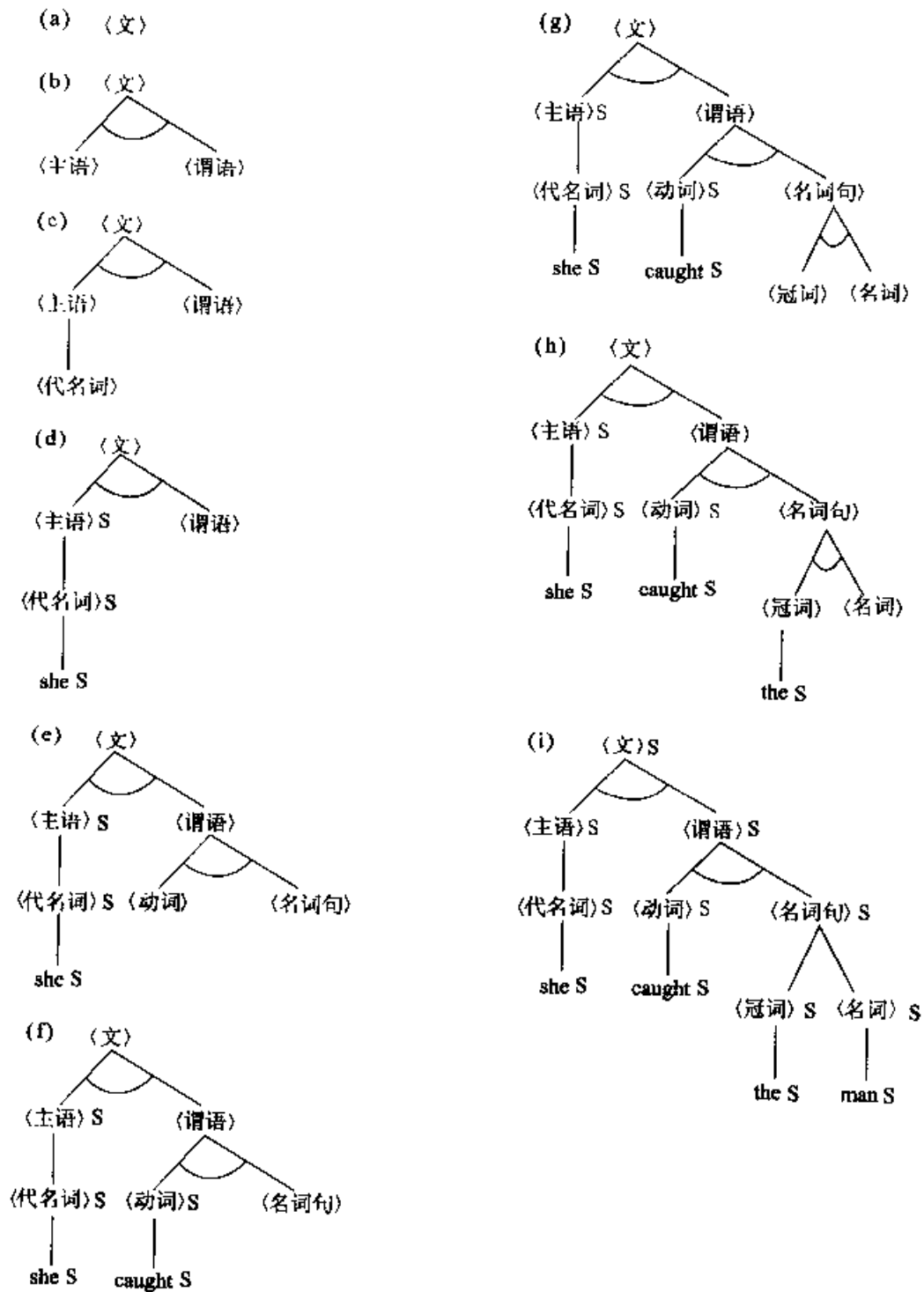


图 2.16 未解图的成长过程

2.5 博弈树的搜索

在象棋和国际象棋等这些由两人进行的比赛中,两个棋手通

过交替地走棋,进行比赛。在这种2人比赛中,确定下一步的走棋问题,也可以用图搜索的方法进行处理。虽然2人比赛的最终结果,可能是赢、输和平局中的一种,但是作为棋手,当然要选择自己能够取胜的棋术。另一方面,对于下一步的棋局,对手要选择使我方变成输局的棋术,对于对手的这种选择,我方棋手是无法干预的,这也是2人比赛的一种特征。

作为具体例子,我们选取图2.17所示的摆三子的终盘局面(a)。这里○和×在结束前各有三步棋可以走,而且设走第一步的是○。这时存在着三个空格A,B,C,应该把棋子放到哪一格内是需要进行判断的难点问题。如果选择在空格A上,则棋盘局面变成(b)。此后应轮到×走棋,这时可供选择的分枝是剩余的B和C。如果这时×选择B,则变成平局,如果选择C,则×成为赢局。在这种情况下,×当然会选择C,因此局面(b)被束缚在输局上。

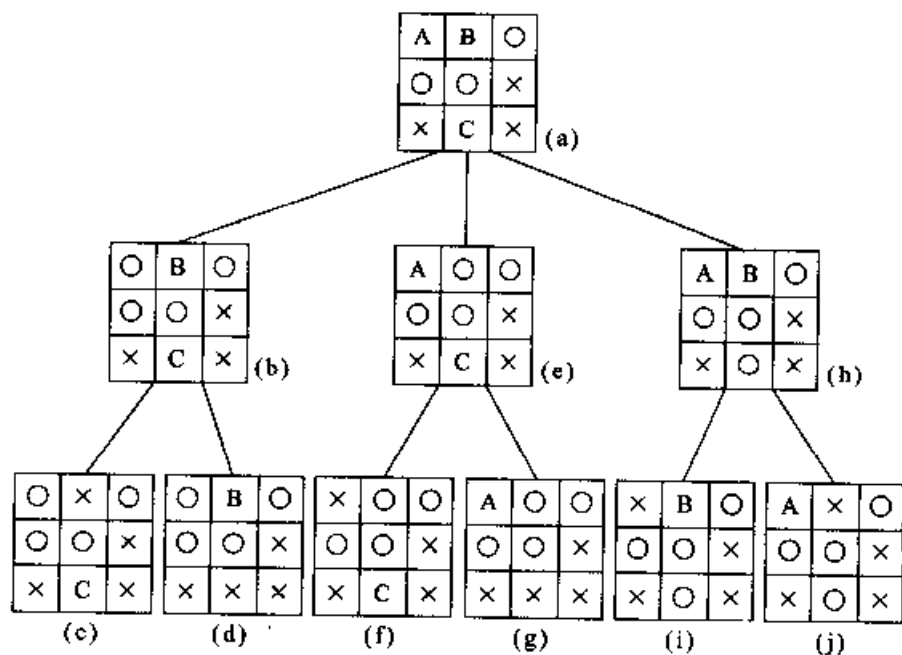


图 2.17 摆三子

其次,我们来探讨○选择B时的局面(e)。这时×的可选择分枝是剩余的A和C。当×选择A时,○会出现两个并排的局面,虽然赢的可能性仍然存在,但是当×选择C时,却能够确保×的赢局。因此,这时×当然也会选择C,从而把局面(e)束缚在输局上。

最后,我们来探讨○选择C时的局面(h)。这时×的选择分枝是剩余的A和B。当选择A时,○会出现两个并排局面,当×选

择 B 时,出现了平局局面。因此,在这种情况下,因为 \times 选择了 B,所以局面(h)也只能变成平局。

综合上述分析可以看出,对于在局面(a)中的 \bigcirc 来说,最好的选择分枝,是将符号 \bigcirc 放在 C 上,这时可以导致平局局面。

在进行这种比赛时,在交替走棋的棋手中,任何棋手都要对后面的状态(局面)进行提前考虑,并且以各种状态的评估值为基础作出最好的走棋选择。状态评估值的给出方法因问题而异,在摆三子的情况下,赢的评估值设为 $+\infty$,输的评估值设为 $-\infty$,平局的评估值设为 0,此外根据与赢局相关连的棋子数目,可以设为 0, 1, 2。而且这种探讨过程,可以用图 2.18 所示的博弈树表示出 MAX 节点与 MIN 节点交替出现的情况。用 \bigcirc 表示的 MAX 节点是从那些子节点的评估值中选取的具有最大值的节点,它表示了自己的棋步。另外,用 \square 表示的 MIN 节点是从那些子节点的评估值中选取的具有最小值的节点,它表示了对手的棋步。反复进行这种评估,就可以得到各个节点的评估值。这种确定棋步的方法,称为极大极小法。

在图 2.18 所示的博弈树中,棋手在这时的最好棋步可以判断如下。图中 a, c, d, f, g, i, j 为 MAX 节点, b, e, h 为 MIN 节点。对于节点 b,棋手选择 d(评估值为 $-\infty$),对于节点 e,选择节点 g(评估值为 $-\infty$),对于节点 h,选择节点 j(评估值为 0)。因此,对于节点 a,显然选择 h(评估值为 0)是最好的棋步。

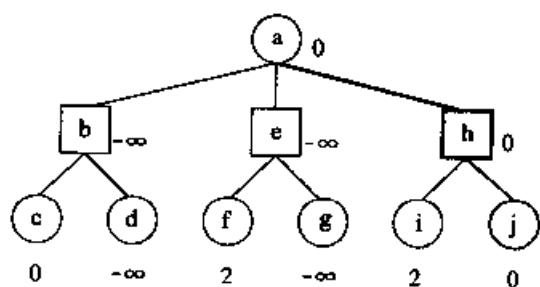
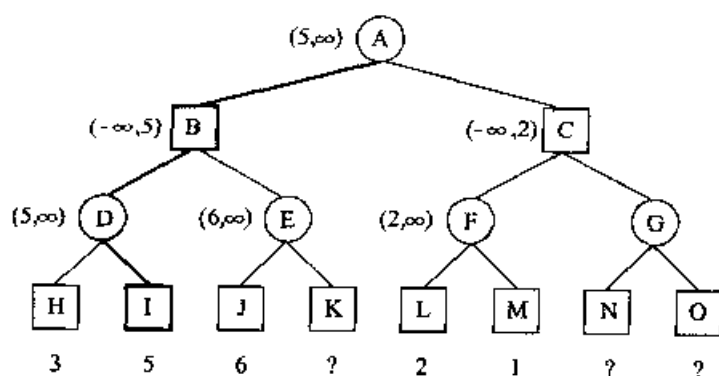


图 2.18 博弈树

不过在极大极小法中,必须求出所有端点的评估值,当预先考虑的棋步比较多时,计算量会大大增加。因此, α - β 方法是一种效率比较高的方法。在 α - β 方法中,采用了两个变量 α 和 β ,它们是最最终可以获得的对最大评估值和最小评估值的估计值。下面通过具体例子说明这种方法。

设给出了图 2.19 所示的博弈树。首先求出 H 和 I 的评估值。这时,因为 MAX 节点 D 上的 5 是确定的,所以在 MIN 节点 B 上确定为 5 以下($\beta=5$)。其次,求得 J 的评估值为 6。因此,MAX 节点 E 的评估值显然为 6 以上。当这样进行时,对于 MIN 节点 B 是不可能选择 E 的。这就确定了与 K 的评估值没有关系,所以没有必要去求它的评估值。这种剪枝,由于节点 E 的评估值超过了它的双亲节点(节点 B),所以被称为 β 剪枝。

图 2.19 α - β 法

这里因为 MIN 节点 B 的值确定为 5,所以显然 MAX 节点 A 的值是 5 以上($\alpha=5$)。接着 L 和 M 的值会被求出来。因为它们分别是 2 和 1,所以 MAX 节点 F 的值可以确定为 2。这时,可以判定 MIN 节点 C 的值为 2 以下。当这样进行下去时,就可以保证 MAX 节点 A 上的值变成 5 以上。这是与 MAX 节点 G 的值没有关系的。因此,也就没有必要去求节点 N 和 O 的评估值了。这种剪枝,由于节点 C 的评估值低于它的双亲节点(节点 A)的 α 值,故被称为 α 剪枝。

计算机棋手战胜国际象棋世界冠军

1997 年,对于人工智能来说,不,应该说对于信息工程来说,都是具有历史意义的年份。那一年,IBM 开发的国际象棋计算机“Deep Blue”战胜了国际象棋世界冠军卡斯帕洛夫。

作为人工智能的分支,国际象棋等比赛,受到了人们更多的关注。在 8×8 的棋盘上,利用双方各 16 个棋子展开的国际象棋比赛,由于其规则明确,所

以可以容易地加装到计算机上,从而通过胜负可以明确地判断出计算机的好坏。所谓国际象棋中的强手,表示了这个棋手具有优越的智能,如果计算机向这种棋手进行挑战并且战胜他,那么这就说明计算机具备了优越的智能。

伴随着计算机技术的进步,国际象棋计算机的性能也得到了提高。1969年曾经开发出了 MACHACK-6,这是一种被称为 C 级的计算机,它在智能上是一种相当于“人类平均”水平的计算机。而在 1981 年,贝尔研究所的两位研究人员开发出了名人级的计算机,进而于 1988 年,卡内基梅隆大学的五位研究生开发出了作为“Deep Blue”前身的“Deep Thought”。这种计算机达到了超大规模名人(仅次于世界冠军的名人)级水平。

“Deep Thought”1989 年时曾向前面提到的世界冠军卡斯帕洛夫进行过挑战。那时该计算机取得了 0 胜 2 负的战绩,从而以完败告终。而作为它的后继者的“Deep Blue”机,于 1996 年 2 月在美国费城,再次向卡斯帕洛夫发起挑战,并进行了 6 盘比赛。在这次比赛中,Deep Blue 初战告捷,但是最终卡斯帕洛夫反败为胜,结果 Deep Blue 以“1 胜 3 负 2 平”的战绩败下阵来。然而 1997 年 5 月在纽约曼哈顿的世界贸易中心,又进行了第三次比赛,Deep Blue 终于以 2 胜 1 负 3 平的结果赢得了对卡斯帕洛夫的胜利。这次比赛的详情,可以从下列网站中查到:<http://www.research.ibm.com/deepblue>.

练习题

1. 试画出由 2 个圆盘构成的梵塔的状态空间图。
2. 试对图 2.20 所示的状态空间图进行:(a)纵向搜索;(b)横向搜索;(c)均一代价搜索;(d)最佳优先搜索;(e)利用 A* 算法的搜索(图中 A 为初始节点,E 为目标节点,各节点的启发值表示在括号内)。

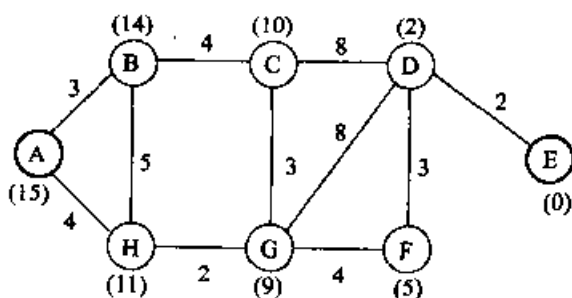


图 2.20

3. 试利用 A* 算法,对图 2.21 所示的状态空间图进行搜索。图中 A 为初始节点,E 为目标节点。另外,试针对启发值的配置情况,阐明有关问题。

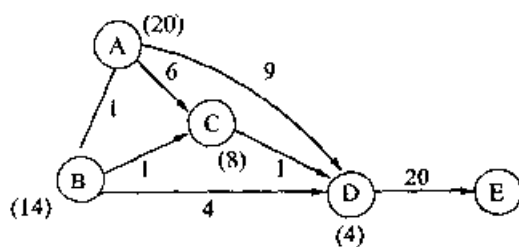


图 2.21

4. 设有由 $\{X \rightarrow ab, X \rightarrow cY, Y \rightarrow a, Y \rightarrow c, Z \rightarrow Yc\}$ 构成的文法规则集合。试利用这些规则将符号列 XYZ 改写成可能的另外的符号列, 用 AND/OR 图予以表示。另外, 试针对目标状态仅为 c 时构成的符号列, 给出表示这个求解过程的解图。
5. 试利用 $\alpha\beta$ 搜索法, 对图 2.22 所示的博弈树进行搜索, 被选择的状态用 \bigcirc 表示, 没有必要进行评估的状态用 \times 表示。
6. 为了求解图 2.1 中表示的 8 数码魔方问题, 试设计出 A* 算法的程序, 并以计算机实施求解过程。在启发函数利用 (a) $h=0$, (b) h 为与位置不同的方砖数, (c) h 为用曼哈顿距离的和等不同情况下, 试对扩展后的状态数进行比较。

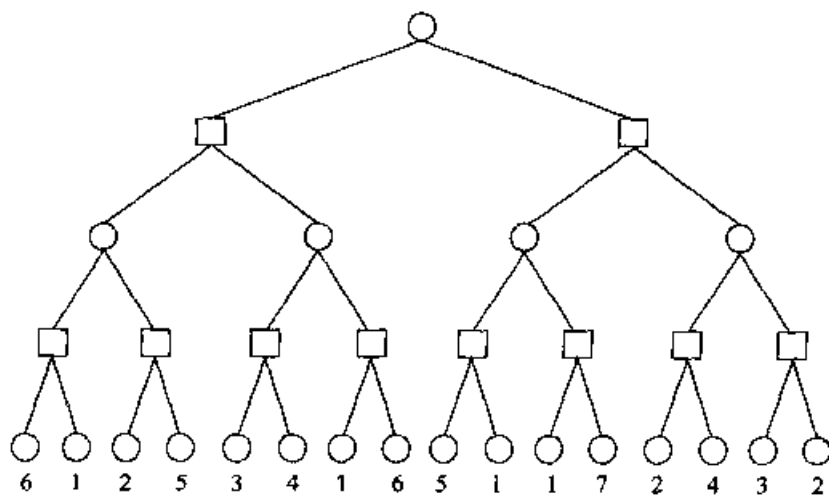


图 2.22

第 3 章

知识表示和推理

人类进行的“聪明”行动,是在对许多已知事实(知识)进行综合,或者说进行加工(推理)的基础上形成的。为了能用计算机实现这种“聪明”行动,怎样表示知识,怎样进行推理就成了需要解决的问题。人类进行的推理是非常复杂的,现在只有一部分推理方法是清楚的。本章将针对在计算机上能够实现的,而且已经实用化了的方法进行介绍。

3.1 知识与推理中的关系

为了用计算机实现人类的智能行动,用计算机能做些什么事情就可以了呢?人类运用已获得的知识认识事物、判断情况,并且设想出一些新事物。即人们要表示知识、处理知识,进而利用知识。为了让计算机去作同样的事情,知识的表示方法以及对其进行处理用的推理方法,就成为必要的了。

因为一般情况下,知识只是以所谓的事实、法则和原理的形式表示出来,所以对它们的处理推理也会变得相当复杂。因此,这里只限于在以实用化为目的的专家系统的处理范围内,对其推理结构进行介绍。

与以前的软件系统比较,专家系统具有下列特点。

1. 处理庞大的未加整理的知识

人类具有数量庞大的知识,但是未必全都经过整理。例如,乘坐电车的所有知识虽然已经公布了,但是能够完全地陈述所有这些知识的人还是不多,不过当提出“要乘坐电车需要做些什么?”,“如何通过自动售票机买票?”之类的一些问题时,多数人是能够回答的。这种对知识进行处理的方法是必要的。从前,对于这种情况,当对知识进行整理但又不能明确地知道处理的流程等时,是不

能开发软件的。

2. 能够与知识的追加、修正和清除进行简单地对应

人类具有的知识,随着对象领域的变化,以及新信息的获得,而相应地发生着变化。例如,当电车的预置模式卡在教育上成为可能时,随之便增加了为使用该预置模式卡所需要的知识。这样,能简单地进行知识追加、修正和清除,就是必要的了。以前的软件,因为每当进行追加、修正和清除时,都必须考虑程序的构造、处理的流程等,这已经成为一项非常困难的作业。

3. 适用于规则和数据具有模糊度的场合

人类具有的知识,不能说全部都是完全正确的。例如,“黑色的鸟是乌鸦”这个知识就不是完全正确的。因为除乌鸦以外,还有其他黑色的鸟和黑色的鸡等。此外,完全黑的鸟是不是存在也还是一个问題。这是一个数据和对数据进行表示时具有模糊度的例子,实际上规则本身也存在有模糊度。对于由某种原因或某一理由产生的现象,虽然多数情况下能够予以正确描述,但是相反的情况却会造成困难。例如,患流行性感冒的人,其症状可以描述为“发烧、咳嗽”等,但是因为“发烧、咳嗽”这种症状也会出现在其他疾病中,所以不能得出一定是流行性感冒的结论。因此,对这种具有模糊度的问題进行处理的方法是必要的。

知识表示与对其进行处理的推理密切相关。在 3.2 节中,我们将对产生式系统的表示法和推理方法进行说明,这些方法在专家系统中得到了有效的应用。产生式系统可以用来应付在上述 1、2 项中遇到的困难。

3.2 产生式系统

3.2.1 产生式系统的构造

产生式系统(production system)是 1973 年由纽厄尔(Newell)提出的,它是用计算机构成的一种系统,这种系统具有模仿人解决问题的行为机构。与人类具有的长期存储器(longterm memory)和短期存储器(short term memory)相对应,产生式系统的记忆场所也采用了两种类型。长期存储器被称为知识库,它是收藏被长期保存的知识的处所。在产生式系统中,将 if-then 规则储备在知识的收藏场所。短期存储器,在作业领域被称为工作存储器

(WM),它是暂时的数据收藏场所。在产生式系统中,由外部给予的数据和从推理中获得的结果将会被记忆。因此,产生式系统基本上将具有图 3.1 所示的构造。

下面我们通过例子对上述系统中的各个部分进行说明。

1. 作业领域

收藏数据(事实的集合)和假设(目标)等。例如,设在动物园中的某个兽笼前,对生活在笼中的动物进行观察。

假设对笼中名字为“太郎”的动物得到了

下列数据。标记在数据前面的 D_x (x 为数字)称为标识符, x 为数据形成时顺序分配的序号。即数字越大,数据越新,这是显而易见的。以下的表示方法,是一种意义容易理解的表示方法,它与在实际的计算机上的表示是不同的。

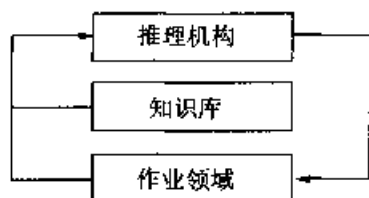


图 3.1 产生式系统的构成

【例】

- D1 (太郎身上有毛)
- D2 (太郎有尖锐的牙齿)
- D3 (太郎有锋利的爪子)
- D4 (太郎身体的颜色是黄褐色)
- D5 (太郎身上有黑色斑点)

2. 知识库

用下面表示的规则的形式把知识存储起来,称之为规则库。条件部是条件文字行列,实行部是实行文字行列。

【形式】

规则名	If	条件部
	Then	实行部

下面表示了一个规则的例子。下面的表示方法也与数据的表示方法一样,它和在实际的计算机上的表示方法是不同的。这里 X 为变量,可以用任意的文字行列代入。规则内相同的变量用相同的值代入,在不同的规则内,即使是相同的变量名,也要用不同的值代入。例如,当把规则 R1 的条件(X 身上有毛)与(太郎身上有毛)相对照时,就会知道应往 X 中代入“太郎”,R1 的实行部变成“追加(太郎是哺乳动物)”。但是,其他规则的 X 的值不受影响。另外,同样的规则也可以再次使用,在其他场合中 X 要用其他的值代入。

在实行文方面,可以利用的操作应针对作业领域进行,“进行追加”和“进行清除”都是可能利用的。

R1	If	(X 身上有毛)
	Then	追加(X 是哺乳动物)
R2	If	(X 喂奶)
	Then	追加(X 是哺乳动物)
R3	If	(X 会飞翔)
		(X 产卵)
	Then	追加(X 是鸟类)
R4	If	(X 有翅膀)
		(X 不是企鹅)
	Then	追加(X 会飞翔)
R5	If	(X 是哺乳动物)
		(X 吃肉)
	Then	追加(X 是食肉动物)
R6	If	(X 是哺乳动物)
		(X 有尖锐的牙齿)
		(X 有锋利的爪子)
	Then	追加(X 是食肉动物)
R7	If	(X 是哺乳动物)
		(X 有蹄子)
	Then	追加(X 是有蹄动物)
R8	If	(X 是食肉动物)
		(X 的身体颜色是黄褐色)
		(X 有黑色条纹)
	Then	追加(X 是老虎)
R9	If	(X 是食肉动物)
		(X 的身体颜色是黄褐色)
		(X 有黑色斑点)
	Then	追加(X 是猎豹)

3. 推理机构

推理机构是产生式系统的一部分,它从知识库中获得满足作业领域表示的状态条件的规则,并从这些规则中选取一种规则予以执行。推理方法大体上可区分为前向推理和后向推理。前向推理是在新的事实被加进数据时,利用知识库中的规则,求出什么样

的事项可以作为结论的这样一种推理方法。另一方面,后向推理则是从想要证明的事项开始,然后利用能使上述事项成为结论的规则和数据,试探地进行证明的一种推理方法。下面针对这两种推理方法说明推理机构的运行情况。

3.2.2 推理机构的运行

1. 前向推理

前向推理的运行,是直到得到求证的结果以前,或者说直到能被使用的规则用完以前,反复地进行下面表示的认识行动的循环过程:

第1步:匹配(matching)。

求出具有下列条件部的所有规则的集合(冲突集合),该条件部被作业领域内的数据所满足;

第2步:冲突消解(conflict resolution)。

根据选择标准,从冲突集合中选择一个特定的规则;

第3步:进行推理(action)。

实现被选择规则的结论部,更新作业领域的内容。

2. 冲突消解

在产生式系统中,一般来说,根据作业领域的数据满足条件部的规则有多个。但是,对于规则来说,基本上不可能同时对几个进行处理,所以有必要确定使用哪一个规则。为此,首先要构成冲突集合,然后考虑从集合中选择一个规则的方法。所谓冲突集合,就是人们所说的规则与满足其条件部的作业领域的数据组集合。另外,从这种冲突集合中选择一个规则的方法称为冲突消解。为了进行冲突消解所采用的方针称为冲突消解策略。冲突消解中使用的信息是规则的条件部与数据的关系。下面我们来介绍在确定冲突消解时,经常使用的一些要素。

- 规则的重要程度 给每个规则标注上重要程度,重要程度高的规则优先进行选择。

- 规则条件部的详细程度 条件部的描述(条件文字)多的规则(详细的规则),优先进行选择。

- 规则的使用时刻 考虑现在时刻与最后被使用时刻之差(未使用的时间)。

- 差值大的规则优先 尽量不使用同一规则。

- 差值小的规则优先 有益的规则要多使用。

- 数据的生成时刻 考虑被作业领域追加的时刻。新规

则优先的情况居多。

现在来介绍使用上述要素实施冲突消解策略的例子。根据编号顺序进行操作,在规则变为一个的时刻,结束处理。另外,在(1)中冲突集合变空时,结束处理。

【LEX 策略 (lexicographic sort)】

- (a) 把已经执行过的一组规则从冲突集合中清除。
- (b) 选择具有更新数据的一组规则。
- (c) 选择规则条件部详细程度大的一组规则。
- (d) 选择任意一组规则。

这里在(c)中提到的条件部的详细程度大,约定为条件部文字的数量多。

现在我们利用上述数据和规则,对前向推理的运行进行说明。这里我们来考虑动物园中兽笼前存在的状况。假设我们看到了笼中的动物。但是很不幸,在兽笼子中没有设置有关笼中动物的说明。于是我们给笼中的动物起了个名字“太郎”,并对其进行观察。设已经得到前面的数据 D1~D5。现在让我们利用这些数据和规则 R1~R9,分析太郎是哪一种动物。

根据数据,能满足条件部的规则只能是规则 R1。即冲突集合只能是 R1 与 D1 的组合。R1 与数据 D1(太郎身上有毛)相适合,所以 $X = \text{太郎}$ 。因此,把 R1 的结论部(太郎是哺乳动物)作为数据 D6 加入到作业领域。至此,最初的匹配、冲突消解和执行的循环过程结束。在产生式系统中,紧接着就要进入更深入一步的认识行动循环过程。在这个循环中,将规则 R6 和数据 D2, D3, D6 的组合加进最初的组合中。当选择 R6 的组合时(太郎是食肉动物)的这一数据作为 D7 被追加进作业领域。这样,在下一个循环中,规则 R6 和数据 D7, D4, D5 的组合也将被追加到作业领域,如果选择规则 R9 的组合,则其结论(太郎是猎豹)被加进作业领域,于是得到求解结果。上述运行过程可以归纳如下:

循环	冲突集合	使用规则	得到的数据
1	{(R1, {D1})}	R1	D6(太郎是哺乳动物)
2	{R1, {D1}}, {R6, {D2, D3, D6}}	R6	D7(太郎是食肉动物)
3	{(R1, {D1})}, {R6, {D2, D3, D6}}, {R9, {D4, D5, D7}}	R9	D8(太郎是猎豹)

3. 后向推理

后向推理是检查给出假设后,能否用数据和规则进行说明的一种推理。在能对假设完全说明(能验证)之前,重复进行下列循环过程。在下列循环中,采用 AND/OR 树进行推理,能同时满足的假设被设为 AND 关系,用来说明某一假设的候补规则被设为 OR 关系。候补规则中条件部的条件被设成了 AND 关系,并且以树的形式表示了出来。最初,能验证的假设是作为 AND/OR 树的根(root)给出的。

第 1 步:匹配(matching)。

从 AND/OR 树的叶中选择一种未经验证的描述,并将其设为假设。求在结论部中具有 AND 假设适应的描述的规则集合(冲突集合),将各规则的条件部的条件设为与关系,并将各个与关系树设为或关系,作为假设的子假设增加进去。

如果找不到规则,则这种假设就不能进行验证。因此,由于以这种假设的验证为惟一依存的父假设(如果不能验证这种假设,则不能说明父假设)已变得不再需要,所以可以从 AND/OR 树中清除。受到这种清除影响的假设也被清除。

第 2 步:规则的选择(rule selection)。

从 AND/OR 树中选择有叶的 AND 关系,即选择一种规则。如果没有能被选择的规则,则推理是失败的。

第 3 步:验证(verification)。

我们来检验被选规则的条件部中记述的条件,即 AND 关系中的子成分是否满足作业领域的数据。满足的记述条件,即被得到验证。AND 关系中的所有子成分如果全被验证,则其父成分也就被验证。即如果条件部的所有记述均被满足,则被选规则的结论部就得到了验证。这时,其父成分和存在于与关系中的所有假设如果全被验证,则这些子假设的父假设也就得到了验证。按这种方式进行,如果 AND/OR 树的根得到验证,则推理即获得成功。

利用在前向推理中用过的同一例子,说明这时的运行情况。最初给出的假设是“太郎是猎豹”。与推理相关的所有内容都在图 3.2 上的 AND/OR 树中表示了出来。另外,虚线表示存在于作业领域的记述,实线表示最终得到的 AND/OR 树。

只有 R9 才是在结论部中具有适合于假设的规则。在规则的条件部中,有三个条件,即“太郎是食肉动物”,“太郎的身体颜色是黄褐色”和“太郎身上有黑色斑点”。在验证的步骤中,基于对作业

领域的检验可以得知第二项和第三项条件满足数据 D4 和 D5, 只有第一项条件不能得到满足。

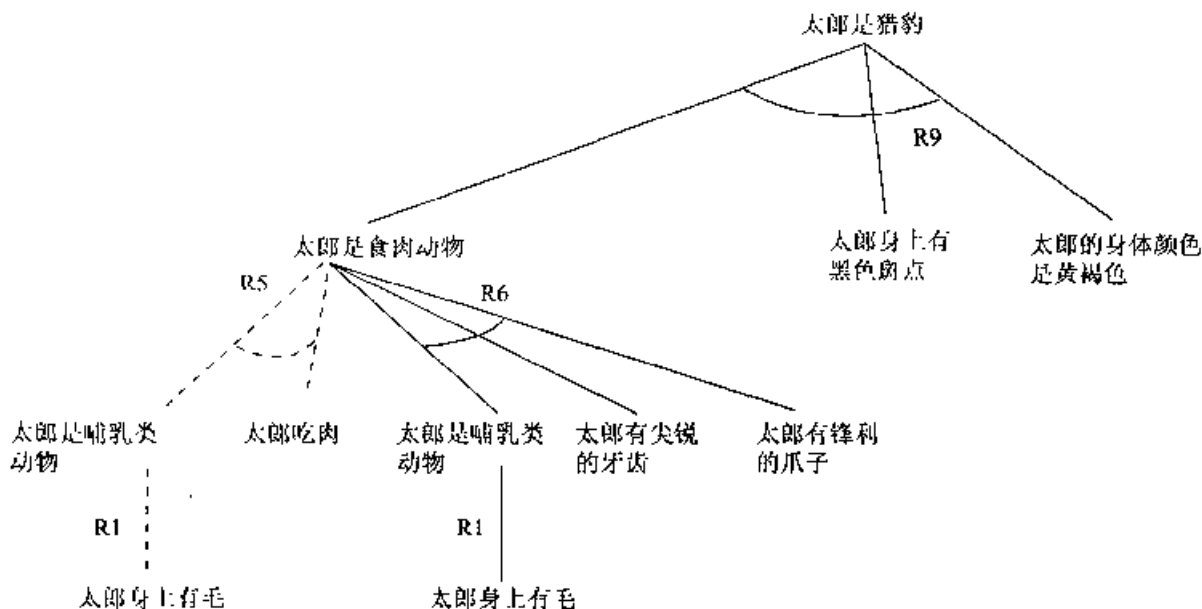


图 3.2 后向推理的 AND/OR 树

在下一个循环中,寻找结论部中具有适合于“太郎是食肉动物”的记述的规则。这时,选择了规则 R5 和 R6,并且作为 OR 关系被添加到 AND/OR 树中。在规则的选择中,假设选定了 R5。能用来进行检验的数据这时为“太郎是哺乳动物”和“太郎吃肉”。因为这两个数据在作业领域都没有,所以可以作为 AND 关系添加到 AND/OR 树中。

其次,作为假设,假定选择了“太郎是哺乳动物”。在结论部中,具有适合于这种假设的记述的规则是 R1 和 R2。设这两个规则中各自的条件构成“与”关系,然后又把它们作为“或”关系添加到与/或(AND/OR)树中。设在规则的选择步骤中,选定了 R1。R1 的条件是“太郎身上有毛”,这个条件存在于作业领域,所以得到验证。因此,“太郎是哺乳动物”也得到了验证。但是,R5 的另一个条件(太郎吃肉)因为未能得到验证,所以仍然予以保留。

然后设“太郎吃肉”为假设。因为找不到在结论部中具有这种记述的规则,所以未能验证。因此,由于在“太郎是食肉动物”的验证中不能予以使用,所以可以从与/或树中清除。在规则的选择步骤中,R6 被选中。这项规则的条件部由“太郎是哺乳动物”,“太郎有尖锐的牙齿”和“太郎有锋利的爪子”组成。在验证步骤中,根据

对作业领域的检验,可以得知第二项和第三项条件满足数据 D2 和 D3。只有第一项条件不能得到满足。

在下面的循环中,选择了“太郎是哺乳动物”作为假设。在结论部中,具有适合于这种假设的记述的规则是 R1 和 R2。在规则的选择步骤中,若假设选定了 R1,则与前面的讨论相同,“太郎是哺乳动物”也得到验证。据此,R6 的条件也满足,于是其结论(太郎是食肉动物)也得到了验证。因为最后的这个条件被满足,所以最初的假设(太郎是猎豹)就得到了验证。

3.2.3 理由(Why)和方法(How)

表示由推理得到的数据与规则之间关系的图称为推理网络。在图 3.3 中,表示了在前一节求出的推理网络。利用这个推理网络,可以与对该推理内容的质问相对应。这里介绍的方法,对前向推理和后向推理两方面都能够适用。另外,为了使这种方法在推理过程中也能应用,系统能够一面与使用者进行对话,一面在推理进行的情况下回答使用者的质问。

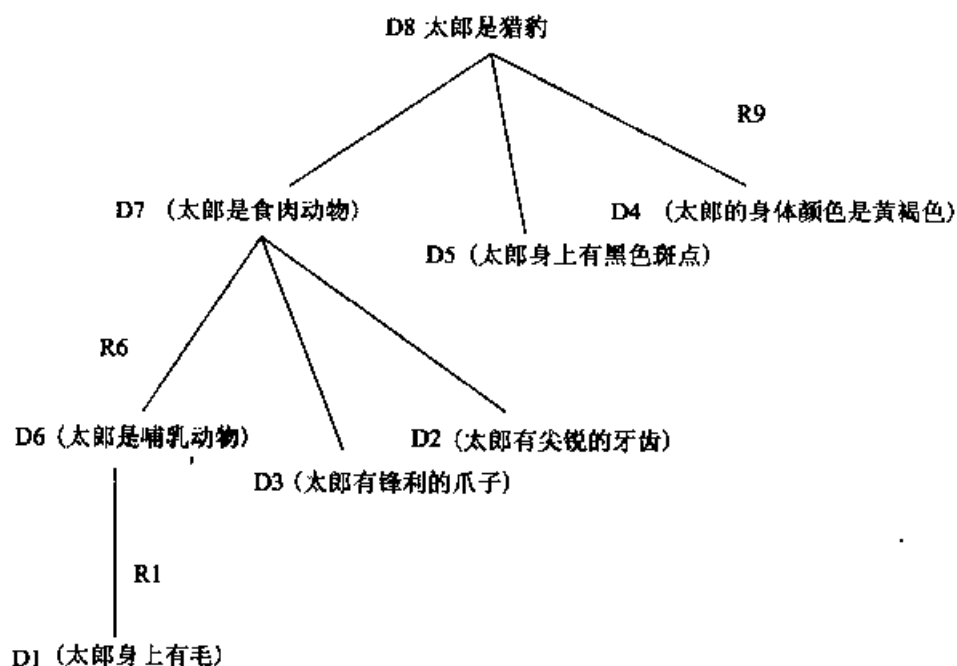


图 3.3 推理网络举例

1. 为什么需要这些数据?

对于表示出的结论,推理系统能够借助于推理网络,根据从其数据向假设方向的追踪,以及在规则的情况下,根据从条件向结论

方向的追踪,说明某个数据的必要程度。例如,针对“为什么检验‘太郎是食肉动物’是必要的?”这样一个问题,根据从“太郎是食肉动物”开始到假设方向的追踪检验,即对规则 R9 的检验,可以构成下列回答:

“‘太郎身体的颜色是黄褐色’和‘太郎身上有黑色斑点’是明确的。根据规则 R9,如果明确了‘太郎是食肉动物’,则一定能判定太郎是猎豹。”

2. 可以得到怎样的结论?

对于生成的结论,推理系统能够借助于推理网络,根据从其根部向叶部方向的追踪,以及在规则的情况下,根据从结论向条件方向的追踪,说明是怎样进行推理的。例如,针对“怎样做才能判定‘太郎是食肉动物’”这个问题,可以根据从“太郎是食肉动物”到数据方向的检验,即对规则 R1 和 R6 的检验,构成下列回答。

“首先,由数据‘太郎身上有毛’和规则 R1 可以得知‘太郎是哺乳动物’。而且根据规则 R1 的结论,和数据‘太郎有尖锐的牙齿’,‘太郎有锋利的爪子’,以及规则 R6,可以得知‘太郎是食肉动物’”。

3.2.4 产生式系统的特征

针对产生式系统的知识性质,该系统具有下列有效特征。

1. 适于作业领域状况的规则的选择

产生式系统自动地选择适于作业领域状况的规则,并且用其中人们认为最适当的规则进行推理。因此,没有必要考虑规则的顺序和关系,适于处理未经整理的庞大的知识。

2. 规则相互独立

规则只是表示条件和结论间的关系,对于其他的规则只能直接利用,无法对它们有区别地进行称谓。因此,在以前的程序进行变更时,没有必要考虑处理与程序中的场所的关系。因为对规则的追加、修正和清除可以自由地进行,所以能够简单地与知识的追加、修正和清除相对应。

3.3 框 架

在 3.2 节中,作为因知识庞大而不易整理的知识的一种处理

方法,介绍了产生式系统。另一方面,也存在着经过巧妙整理的知识。即也存在着典型形式的知识(典型的知识)和表示层次关系的知识(阶层知识)。进而存在着与依附于事态的运行和处理相关的知识(程序式的知识)。框架就是为表示这样的知识而提出的一种方法。

图 3.4 表示了为实现框架概念的一种构造例子。这种表示是一种用图表示的直观上容易理解的框架,不是在计算机上的实际表示。欲表示的事态作为框架名记述,与事态有关的属性作为槽名进行表示,对属性的知识,在槽中作了记述。侧面表示了对数据进行规定的信息。例如,数据表示事实、假想事态、程序的启动方法等。

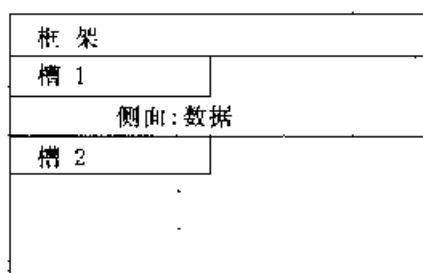


图 3.4 实现框架的例子

框 架

框架是明斯基(Minsky)1975年提出的一种知识表示方法,它是作为理解人类的记忆构造和推理的机构(machanism)而被研究出来的。这种框架理论,在与人类的记忆和推理机构有关的认知心理学方面,具有创新意义,是一种体系化了的表示方法。而且众所周知,为了进行知识表示,具有许多优良特征的框架表示方法,能够适用于图像理解和自然语言理解等广泛的领域。

在诸如“进入起居室啦”,“去参加孩子的生日聚会啦”这样一些类型的情况中,框架是一种表现期待状况的数据构造。在每个框架中,都添加有各种各样的信息。在这些信息中,假设有与框架的应用方法有关的内容,那么接着也会有与可能发生和期待的内容有关的信息,另外还有一些内容,是有关在实现这些期待时,能实施的一些行动方面的信息。

明斯基关于框架的观点表明,这种方法有助于说明各个领域内与人类的智能有关的许多现象。框架观点本身并没有什么特别的新内容,因为这些内容在巴特利特(Bartlett)的模式(schema)中和库恩(Kuhn)的范例(paradigm)中已经存在。但是,当明斯基的框架理论使计算机具有了智能行为时,对于人工智能研究者来说,却产生了非常大的影响。

3.3.1 典型知识与框架

作为典型知识的表示的例子,在图 3.5 中示出了一种框架的

例子,这个框架表示了一次以人工智能系统开发为主题的一种人工智能(AI)会议。通常,会议的必要信息是时间、会议地址、会议目的和会议的出席者,在图 3.5 所示框架中,记述了这些内容,作为记述场所的槽(slot)应予以关注。这样一来,对于某一个事项,一般都能够简单地定义出它的必要情况和属性,从而可供人们参阅。

会议37的框架	
时 间	
值:	2000年9月11日
地 点	
值:	AI专题研讨会会议室
目 的	
值:	人工智能系统开发
出席者	
值:	佐藤, 山田, 安部

图 3.5 会议 37 的框架

因为框架在上述类型中表示的是期待的状况,所以当考虑的有关人类的行动和对象改变时,也应变换成其他类型的框架。现在以三角锥为例进行说明。图 3.6(a)中表示了一种能够看到的三角锥的情景。当观察点向这个三角锥的上方移动时,看到的情景将如图 3.6(b)所示。三角锥及其被看到情景的框架表示于图 3.7 中。若从正面看三角锥,则看到的情景像(a)那样,若从正上方看三角锥,则看到的情景像(b)那样。在看到的情景(a)中,只能看到三角形 A 和 B, C 被隐藏起来。在表示两种看到的情景的框架时,三角形 A, B, C 作为值,是共有的。但是,被分配的槽是不同的。这种方法表示了同一物体从不同角度看到的情景,而从某一框架向另一框架的变换表示了观察点的变换。

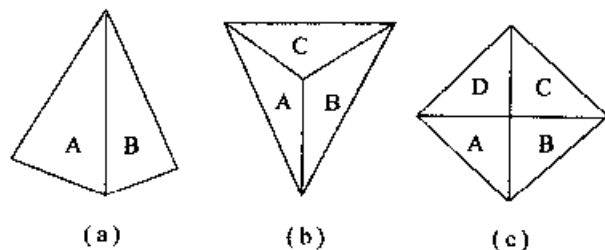


图 3.6 三角锥和四角锥被看到情景示例

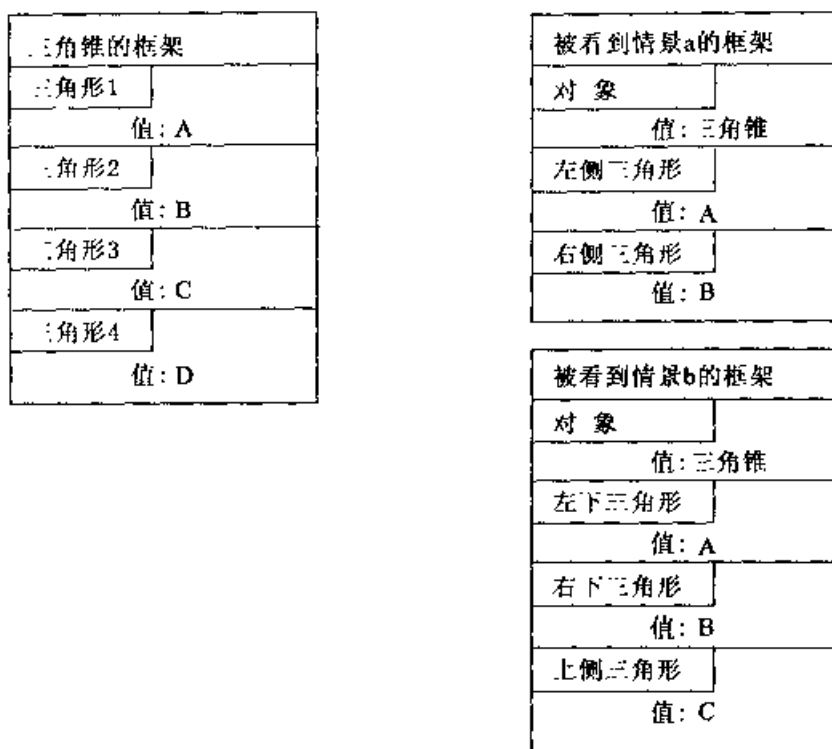


图 3.7 三角锥与被看到的情景

我们观看物体时,设被看到的物体如图 3.6(a)所示。因为适合于三角锥的被看到的情景(a)的框架,所以假设判断这个物体为三角锥。因为三角形 C 被隐藏,所以看不见。当把观察点向物体的上方移动时,看到的情景如果与图 3.6(b)相匹配,则上述假设就会变成正确的判断。但是,如果出现了图 3.6(c)那种情景,则与三角锥被看到的情景(b)框架的匹配就会失败。在这种匹配失败的情况下,根据信息检索网络,依照三角锥的情况,进一步选择出适合于情景的框架,并予以应用。在这种情况下,与四角锥相关的框架得到了应用。

3.3.2 阶层知识与特征的继承

知识不只是具体的事项,由于在知识中汇集着事物的共同性质,所以它是以抽象化了的事物存在的。框架适于表示具有分类学阶层构造的概念。对于这种阶层构造,上层框架其有的特征信息是与该框架相联系的所有下层框架共用的。即框架能够继承其上层框架的信息。因此,使得简洁的表示成为可能。现在我们来考虑关于会议的问题。设实际上被召开的一次会议为会议 37。会议 37 是一次人工智能(AI)会议。于是会议 37 与 AI 会议中的

重复性特征,没有必要在会议 37 中记述。在图 3.8 中,分别表示了会议 37 和 AI 会议的框架。种类槽是表示框架上下层关系的一种手段。

AI会议的框架	
种 类	
值:	一般会议
时 间	
值:	每周星期五
地 点	
隐含值:	AI专题研讨会会议室
目 的	
值:	人工智能系统开发
出席者	
隐含值:	佐藤, 山田, 铃木

会议37的框架	
种 类	
值:	AI会议
出席者	
值:	佐藤, 山田, 安部

图 3.8 表示 AI 会议与会议 37 的框架

PA 会议框架是一般会议框架的下层框架。在 AI 会议框架中,记述了召开会议的主要信息(时间、地点、目的、出席者)。这个会议的目的是讨论有关人工智能系统的开发问题,它记载在目的槽内。在时间、地点和出席者的诸槽内,如果没有涉及到相应的值,则可以使用记述的默认值。例如,若会议的地点没有特别指出,则意味在 AI 专题研讨会会议室举行。

图 3.8 中的会议 37 可以用两个槽表示。这时只在种类槽内记述了它的上位框架是 AI 会议,而在出席者槽内记述了出席者是佐藤、山田和安部。不过,对于框架系统而言,由于它继承了上位框架的特征,所以可以从 AI 会议得到它的时间、地点和会议的目的值。虽然在 AI 会议中出席者的默认值为佐藤、山田和铃木,但是在会议 37 中,显然由安部取代了铃木出席会议。这是一个表示了具有例外情况的记述例子。

3.3.3 程序知识及其启动

用框架表示静态事实时,一般都表示成宣言的结构形式。但是,当需要进行推理和求解问题时,就会把程序附加在槽内。下面以图 3.9 中表示的一般会议为例进行说明。

在一般会议的框架中,采用了 IF-NEEDED 和 IF-ADDED 等表达式。所谓表达式,是指当满足一定的条件时能自动地被启动的程序。当向对应的槽进行访问时,这些表达式会被启动。在一般会议的下部层面中,会设定会议出席者的获取方式,这时如果没有这个值,则启动 IF-NEEDED 侧面中表示的程序 ASK,对使用者进行质问,从而得到出席者的值。在下部层面中,当在时间槽中分配了值时,IF-ADDED 侧面中表示的程序 ADD-TO-CALENDAR 会被启动,从而把时间填写到日历中。

一般会议的框架	
种 类	值: 社会活动
时 间	IF-ADDED:(ADD-TO-CALENDAR) (填写日历的程序)
出席者	要求: 有关人类社会的事情 IF-NEEDED:(ASK) (质问使用者的程序)

图 3.9 一般会议框架举例

3.3.4 框架的特征

典型知识, 阶层知识和程序式知识, 通过精心整理后, 可以用一种结构(框架)来描述。

本体论

一般来说, 本体论(ontology)是哲学上采用的术语, 被称为“存在论(与存在相关的体系理论)”。若从知识表示的观点看, 可以说“它确切地揭示了问题领域内的概念与关系, 并且给出了具有明确意义的定义”。简单地说, 它揭示了表示问题领域所需词汇的意义, 和它们之间的关系与制约, 并且根据给

出的表示形式,可以对对象进行描述。例如, size 是表示范围大小的谓词, small 给出了表示范围大小度量标准的定义。而且根据给出的一阶谓词逻辑式的形式,使得 $\text{size}(x, \text{small})$ (x 的范围大小是小的) 这种表示形式成为可能。

人类与其伙伴,或者说人类与计算机在智能性工作中进行协同作业时,知识是共有的,因此在知识表示的意义上必须是通用化的,为了做到这一点,本体论就变得非常重要了。

因为本体论基本上只与问题领域有关,所以只有问题领域的数目存在。但是,为了进行真正的智能判断,有必要综合各个领域的知识,为此,正在进行通用的本体论的研究工作。

练习题

1. 试述产生式系统的特征。
2. 对于产生式系统,当把下列数据加进作业领域时:
 - D1 (太郎身上有毛)
 - D2 (太郎吃肉)
 - D3 (太郎身体的颜色是黄褐色)
 - D4 (太郎身上有黑色斑点)
 - (1) 作为冲突消解策略采用了 LEX 策略后,试问前向推理应如何运作? 并表示出各个循环中的冲突集合、被选定的规则(使用的规则),以及得到的数据。
 - (2) 示出图 3.3 那样的推理网络。
3. 利用图 3.3 中的推理网络,回答下列问题:
 - (1) 为什么有必要调查“太郎是哺乳动物”?
 - (2) 怎样才能判定“太郎是哺乳动物”?
4. 试说明用框架可以表示什么样的知识。
5. 名称为会议 38 的会议,于 2000 年 10 月 2 日在信息会议室举行。出席人员是佐藤、山田、铃木等三人,已知讨论的议题是人工智能的开发。试考虑会议 38 框架的构成。
 - (1) 试用惟一的框架表示。
 - (2) 若利用图 3.8 中 AI 会议的框架结构,试问会议 38 的框架会变成什么形式?

第 4 章

机器学习

人们已经充分地认识到,人与机器之间的决定性差别是有无学习能力。让处于发展中的机器去完成人类那样的学习,也许还只是一种梦想。在机器中,利用事先编好的程序完成上述任务,事实上目前也做不到。可是,学习本身可以编成程序却是另外一回事,许多研究者的工作证实了这种可编程的可能性。这种研究领域被称为机器学习(machine learning)。

本章首先介绍机器学习的定义和研究历史,以及机器学习的分类,然后介绍过去已经提出来的几个基本概念和学习方式。

4.1 关于学习和机器学习

4.1.1 什么是学习

简单地说,所谓学习就是现有的各种各样形式的学习。一般来说,我们所说的学习到底是什么呢?现在我们来举几个例子。

【学会技能】

- 婴儿学会爬行和用两脚走路。
- 学会骑自行车。

【提高精度】

- 成为三发一中的优秀射手。
- 高尔夫击球手变成为单击球手。

【提高速度和耐久力】

- 达到在键盘上每分钟输入 120 个字的速度。
- 能进行 50 米自由泳。

【增加知识】

- 死记硬背了 5000 个英文单词。
- 通过在国外生活,变得能够理解当地的语言。

- 变得能够区别颜色和形状。
- 能理解复数的概念。
- 学会求解“鸡兔同笼算法”的问题。

首先,让我们追溯到幼儿时期,作为一种反复实践的结果,我们应该能学会用双脚走路的技能,同时也能够通过思维,学会绘画与写字的本领(这种学习称为自主学习或称为基于行动的学习)。从父母指点红色的邮筒和苹果等红色物体中,人们获得了有关红色的概念(通过例子的归纳学习)。相反,如果提供的是非同一般的建议,则应牢牢记住,这时应对那些听起来属于言外之意的内容,进行加工整理,从而给出确切的含义(基于建议的学习)。另外,对于没有记忆经验的人来说,死记硬背九九表歌、英语单词和年表,是一件很不容易的事情(死记学习)。总之,这些全都是学习。再者,利用积累的经验,可以避免走盲目思索的弯路,如果进行的是训练人们具备灵活判断能力的学习(效率化学习),那么通过观察现象,寻找出适于这种情况的法则,这时就称为高级推理学习(发现的学习)。包含在这样一些学习范畴内的行为是非常广泛的。

总结上述内容,我们不能不得出下列关于学习的定义:“所谓学习,乃是人类“和”问题求解系统等的媒体(agent),通过与环境进行对话,为解决将来的问题进行准备,并且在某些意义上,指的也是全面提高自身能力的行为。”这种学习的主体,即媒体为机器时,则称这种学习为机器学习。

在学习中,分为有教师的情况(有教师学习)和无教师的情况(无教师学习)。在前一种情况中,教师和学生分别为自主的媒体,因此构成多媒体系统。机器学习当然担负的是学生方面的任务,学习的成败受到教师好坏的左右,这与人类学习的情况是相同的。顺便提一下,以教师方面的媒体为中心进行研究的内容,称为 ITS (intelligent tutoring system)。在前面的定义中写有“与环境进行对话”,在有教师的情况下,相当于与教师进行对话。在无教师的情况下,通过观测自身的行动和发出的声音对外界的作用及其效果,必须自己创造出自身的控制知识(这使我们回想起交通工具的行驶情况,以及打靶比赛等时的情况)。这时确实存在着与环境的对话。这种学习称为自主学习或称为强化学习,自主的行为被看作是智能机器人不可缺少的功能。

不管有没有教师,如果产生或获取了新知识,或者在精度和响

应速度等方面均可以看到比过去有一定程度的提高,从而能够使自身得到完善,那么就可以说媒体进行了学习。如果按照上述学习的定义,计算机程序化了的所有推理,或者即使仅仅是检索结果的存储,在将来解决问题时,如果能使它们的结果得到有效地利用,那么就全是机器学习。

例如,近年来在引人注目的机器学习的结构中,有一种基于说明的学习(explanation based learning, EBL)。这种方法是通过对一个例题的求解取得经验,也就是通过分析获得问题解的推理过程,形成相当于解题捷径的规则,于是就构成了对将来遇到的同类问题的快速求解技巧。如果深入探讨这种考虑方式,人们也许会产生这样的疑问,即对日文假名汉字变换系统的频度文档的利用,以及进一步对硬盘超高速缓冲存储器的利用,是不是也可以称为学习呢?虽然从广义上说,把这些也称为学习并不错,但是从狭义上说,机器学习是指那些伴随有归纳、类推和演绎等一些具有推理的情况,这也是一种普通情况。因为基于说明的学习,伴随着演绎推理和规则的一般化,所以即使是从狭义上说,也会被看作是机器学习。

4.1.2 机器学习的研究历史

在人工智能的范围内,人们注意到,符号机制与连接机制往往是在对立中求得发展的。机器学习也属于这个范畴,实际上存在着所谓符号逻辑方法和神经网络方法两大流派。所谓神经网络方法,就是按照原样模仿脑(神经系统)的构造,并且通过调整神经元结合加权这种形式,构成实现学习机构的机器学习。所谓符号逻辑方法,则是将谓词逻辑和规则等符号表示的知识学习作为目的。

下面对机器学习的历史做一个简单的概述。

用机器进行学习的最早期的研究工作,是罗森布拉特(Rosenblatt)的以感知器(1957年)为代表的模式识别,和以联想为目的的神经网络的学习。实践证明,1969年由明斯基(Minsky)和佩珀特(Papert)提出的感知器,其功能受到限制,因此,神经网络自身的研究开始走入低潮。其后,人工智能领域的全部研究工作,都倒向了符号机制一边,由鲁梅尔哈特(Rumelhart)提出的反向传播学习的应用,消除了上述限制,从而再次使神经网络的研究兴起,也许可以说又到了所谓神经网络时期。因为在第5章中详细地阐述了有关神经网络的问题,所以请读者参阅那里提供的材料。

让我们把目光转向符号论的机器学习。神经网络的研究处于衰落的时期,即从20世纪70年代到80年代这段时期,正是符号机制兴起的时期。这个时期进行的机器学习的研究内容,主要是从事例出发的概念归纳学习。作为初期的研究,温斯顿(Winston)的拱概念学习(1970年)和米切尔(Mitchell)的变型空间法(1977年)等是比较著名的。这些研究构成了归纳学习的基础。作为这一时期后期的研究,有昆兰(Quinlan)的决策树的学习法ID3(1979年),和作为其发展系统的C4.5,以及夏皮罗(Shapiro)的从事例出发的逻辑程序归纳合成系统MIS(model inference system, 1981年)等,这些都是比较著名的。

这个时期还是费根鲍姆(Feigenbaum)倡导的知识工程学的兴盛时期。在这期间,许多专家系统都处在被开发过程中,知识获取的重要性已被人们所认识。知识获取这个术语,主要是用来表示在专家系统中,根据对专家的访问,获得专门知识这样一种事实。在现实情况中,因为知识获取基本上以所谓支援系统形式的研究居多,所以大体上有别于机器学习,但是就研究的动机而言,可以说两者是完全相同的。

前面提到的效率化学习和自主学习,在机器学习领域,是近期(20世纪80年代后半期)研究工作得到蓬勃开展的学习形式。除此之外,作为新课题,提出了计算论学习理论。其学习算法的良好程度,可以考虑用计算论进行评估,作为学习方法的一种具体提案,具有代表性的是瓦良特(Valiant)的PAC学习。在PAC学习中,对学习领域内的概率要素进行了假定,允许被学习的概念与学习目标的概念之间存在差别,并且采用进行近似学习的模型。基于这种情况,显然可以对学习程度、学习效率,从而对学习能力的评估。在为学习而进行的训练事例中,噪声的混入和学习参数的适应控制等,至今在学习理论中尚未考虑,因此,人们期望计算论学习理论能在这方面发挥作用。

4.1.3 机器学习的分类标准

学习可以用各种各样的观点进行分类。作为分类的标准,可以考虑采用下列一些种类:

1. 知识表示方法

问题求解模块的结构是怎样一种形式?在那里用到的知识又是怎样表示的呢?

由于知识表示存在着可能相互变换的情况,所以学习的本质上的分类标准不可能得到,但是在参数表示、神经网络、决策树等方法中,由于在记述上是一种强限制的表示,所以它们各自形成成为一种单独的学习范例。

2. 依靠学习强化对象

用什么样的观点提高媒体的能力呢?

- 增加知识量(增大解决问题的范围);
- 增大响应速度;
- 精练的知识记述;
- 增加正确的回答率。

以前,所谓机器学习与上述第一项中列举的,以增加知识量为目标的研究,几乎是一样的,那时,希望与现实之间存在着相当大的距离。近年来,上述其他各项内容已成为被重视的观点。响应速度的增大,与在限定的时间内,能够处理多少问题这种能力的提高密切相关。同样,在记忆容量受到限制的情况下,能对知识的整理精练到何种程度,是和能与多少种类的问题相适应密切相关的。另外,以前都是把学习没有错误的完全的知识作为前提的,但是实际上允许知识的不完全性,所以便出现了正确的回答率这种观点。

3. 从外界得到的信息

从外界得到什么样的信息? 这些信息又是怎样得到的呢?

首先,在学习中存在

- 有教师的学习(supervised learning);
- 无教师的学习(unsupervised learning)。

虽然教师和学生是具有共同目的的行为媒体,但是哪一方处于主导地位呢?

- 教师给学生提供信息(教师主导型);
- 教师回答学生提出的问题(学生主导型);
- 双方根据需要边提问边讲授(交互型)。

另外,当考虑把教师提供的信息与学习目标的知识进行比较时,有

- 一般化了的情况;
- 特殊化了的情况;
- 提供的正是学习目标的知识本身的情况。

所谓一般化了的情况,比如在零博弈的情况下,提供“请先占据棋盘角落”这样一种建议,就属于这种情况。这种策略在棋赛终

盘时是不适当的,因此学习者应对规则的适用条件作严格的规定,从而有必要对知识进行特殊化处理。反之,对于所谓特殊化了的情况,只有像“这个是椅子”、“那个也是椅子”这样表示的具体例子才是合适的。当对提供给学习者的具体例子进行一般化处理时,必须得到关于什么是椅子的规则的记述。

在没有教师的情况下,学生对外界产生作用行为时,与有教师的情况是不同的,这时无法知道学生的行为是正确的还是错误的,就像人们无法听到天堂的声音那样。但是,作为一种替代方式,可以把成功和失败以自己的愉快感觉或痛苦感觉形式进行反馈,然后以此为基础,就可以确定学习到正确的行动规则。

另外,作为给予(获得)信息的另外一些方式,也可以得到下列分类:

- 根据学习情况,一次次提供信息的场合;
- 一次全部集中提供信息的场合。

例如,4.2.1节中说明的拱学习就属于前一种场合,4.2.2节的决策树的学习就属于后一种场合。

4. 先验知识和偏移

如何提取先验知识与学习结果得到的知识之间的关系呢?这种关系对于能够学习的知识来说,也是一种标准和方针。

一般来说,学习者具备哪方面的先验知识呢?对于获得的信息,学习者应当试一试,看看用自己具有的先验知识能否对其进行说明,并且从不能说明的情况开始,学习相应的补充性知识,这称为知识调节(knowledge accommodation)。另外,由学习而生成的知识与先验知识不发生矛盾,并且对知识的总体进行系统的整理,这是人们最希望做到的。这种对知识库管理上的调整,称为知识同化(knowledge assimilation)。

这样,可以得到的知识或规则应具有的性质,以及对其记述的限制,一般的就称之为偏移(bias)。先验知识也是偏移的一部分。例如,基于说明的学习,是根据领域的知识对给出的具体例子进行说明,并且是在使这个过程一般化的形式上的学习。因此,什么是可以学习的,在很大程度上取决于领域知识,即取决于先验知识。此外,从几个可能导出的适当的候补规则中,选择出若干个特定的规则时,必须有标准(operationality criterion,操作性规范)。领域知识和操作性规范就是基于说明的学习的偏移。

5. 推理(知识获取)方法

应用什么样的推理方法能创造新的知识呢? 当我们应用下述方法时,是可以创造新知识的。

- 演绎推理;
- 归纳推理;
- 构想,类推;
- 发现等等。

当谈到机器学习时,人们会默认,这指的是源于示例的归纳推理时期的方法,历史上人们看到的最多的研究例子,是基于归纳推理的学习。然而归纳推理是一种正确性得不到保证的推理。类推和发现也同样是这样,有关这些方法的完善的理论体系,目前尚未建立起来。与这些方法相对应,演绎推理虽然在逻辑意义上不创造新知识,但却是一种正确性得到了保证的推理方式,而且因为它已进入到自动化阶段,所以有很高的实用价值。

在以下各节中,我们将介绍归纳学习和演绎学习中的研究示例,它们在研究方面都取得了巨大进展。

4.2 应用归纳方法由示例学习概念的定义

当要传达人们的概念时,人们经常看到的作法是拿出具体的例子(样品,例证)给人看。这样一来,通过例子就可以预测这些例子归属的集合总体的性质,我们把这样的学习称之为归纳学习。

对于归纳学习中能用到的概念的表示方法,被区分成两大类。一类是逻辑式和语义网等,一般来说,这是一种表示能力比较高的知识表示方法,另一类是属性表或者决策树等,这是一种在表示上具有强约束的知识表示方法。虽然不能一概而论,但是在采用前一种知识表示的情况中,是依次进行举例说明的,而在后一种情况中,则多半是把举例说明融合在一起进行介绍,并且以此为前提进行知识表示。

在本节中,我们将采用温斯顿的拱学习作为前一种方法的例子,并且采用决策树学习作为后一种方法的代表,分别进行说明。

4.2.1 温斯顿的拱学习

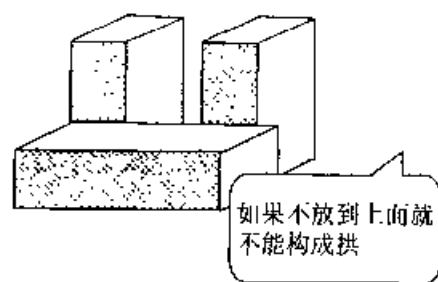
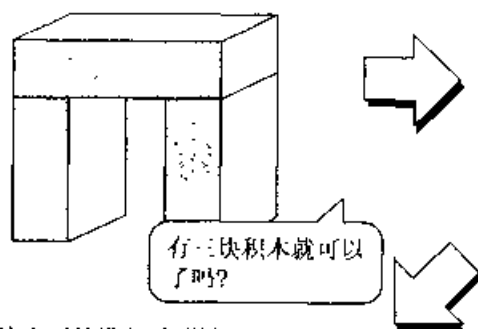
基于温斯顿(Winston)拱概念的归纳学习,对于理解归纳学

习,是非常容易明白的。另外,若基于机器的学习为可能时,那么会暗示这样一种事实,那就是在这种学习中,良好的记述和良好的训练是不可欠缺的,因而它是一种非常完善的例子。

图 4.1 中的①~④,表示了教授拱概念时的示例的情况。这里,①和④分别是构成拱的正确的具体实例,②和③则是不构成拱的正确的具体实例。这时像①和④那样,它们是适合于某一概念的具体例子,称之为正例,反之则称为负例。②和③是负例,它们与正例不同,但显示在书面上则相差无几。在②中,只要把横向比较宽的长方体放到其他两个长方体的上面,②就会变成正例。另外,在③中构成支柱的两个长方体因为紧靠在了一起,所以不能称为拱。这种因在书面上表示的微小差别而未能变成正例的负例,称为近似样品。应用近似样品可以达到这样一种效果,即根据某一概念,它可以清楚地显示出什么是重要的。

①这是拱(正例1)

②这不是拱(近似样品1)



③这也不是拱(近似样品2)

④这是拱(正例2)

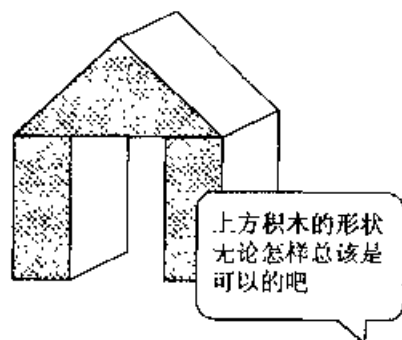
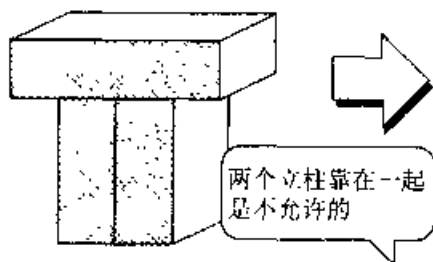


图 4.1 拱学习的学习示例

1. 基于语义网络的积木世界的表示

通过上面的举例,人们可以容易地理解拱到底是什么,那么在用计算机对其进行实现时,怎样做才合适呢?首先必须知道,这实际上是对用计算机处理的对象的描述。

这里让我们考虑用语义网络来描述对象和模型。

设个体、集合和概念为节点,若考虑用连线描述它们的关系,则对于图 4.2(a)中的情景,可以用图 4.2(b)那样的网络(图)描述。在图(b)中表示了下列事实,即积木 1 支撑着积木 2,积木 1 是矩形积木(brick),积木 2 是楔形积木(wedge),它们的总称为积木(block)。与图(a)的情景相当的部分是黑的粗线部分,浅色部分则是对应于事先给出的积木世界的先验知识。下面,为了容易地理解上述说明,我们决定采用图(c)那种描述方法,它省略了表示形状的那部分内容。即,设三角形的节点,与其对应的积木是楔形,四角形的节点,与其对应的积木是矩形。在不涉及积木的形状的情况下,采用椭圆形节点。

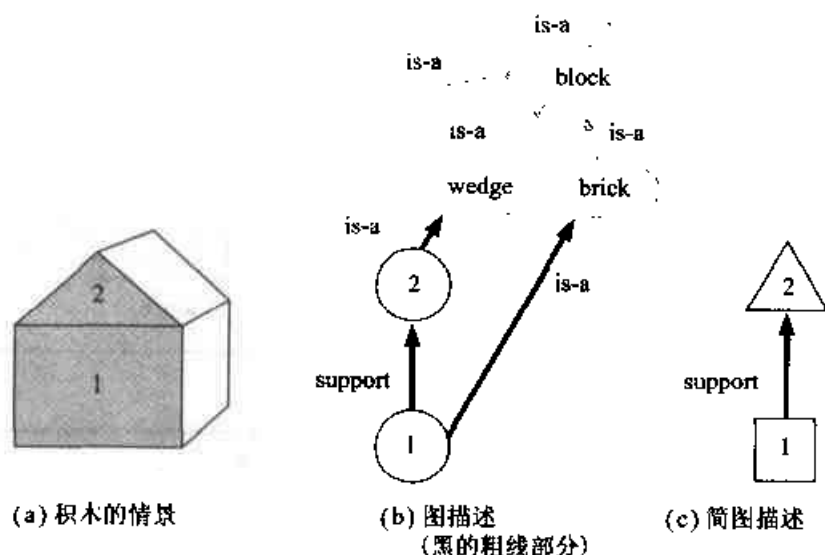


图 4.2 积木情景的网络描述

2. 根据逐次展开的示例生成归纳模型

利用这种描述方法,对前面的拱学习过程进行描述,其结果将如图 4.3 所示。带有边框的描述,相当于图 4.1 中的四个样本。涂有浅灰色背景的描述是拱的模型,通过示例,依次示出了模型描述的严谨形式。

首先,图 4.1①的情景当然是以正例的形式显示了出来,但是因为目前尚无其他信息,所以其网络描述就变成了原始拱的初始候补模型(图 4.3①)。在这个阶段,哪个节点或连线是具有本质意义的呢? 或者有时由于这种信息只存在于具体的例子中,而使问题的答案变为不明确。

其次,图 4.1②中的情景是以负例的形式给出的。如果把②的网络描述(图 4.3②)与模型 1 的描述进行比较,则可以看出,它们的不同表现在是否有 support 连线上。这表明,对于拱来说,两条 support 连线是不可缺少的。为了明确地表示这种情况,support 连线被升格为 must-support 连线,于是产生出了拱模型 2。

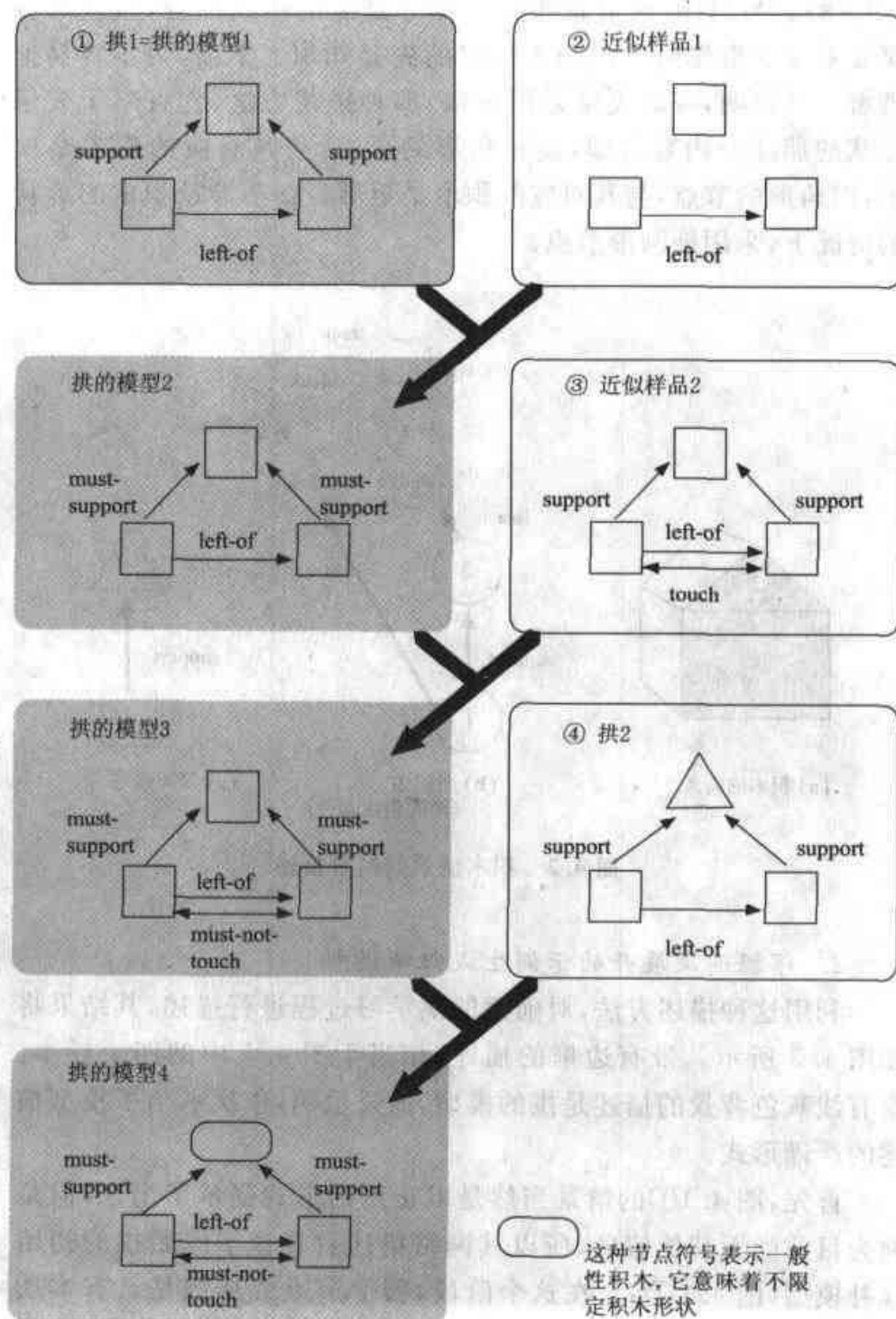


图 4.3 拱学习网络的表示

再者,图 4.1③的情景是以负例的形式加进来的。③的情景满足当时模型 2 的全部必要条件,只有 touch 连线是多余的。③之所以不能成为拱的原因,显然是由于其 touch 连线造成的。因此,在模型中下边两个节点之间,为了表示出不存在 touch 的关系这一事实,特追加了 must-not-touch 连线,从而构成了模型 3。

最后,图 4.1④的情景是以新的正例加进来的。此前,在所有的例子中,被支撑的节点都是矩形积木(brick),而在这个新的例子中,被支撑的积木却变成了楔形积木(wedge)。如果根据所谓的拱概念,这个节点是矩形积木这件事具有本质的意义,那么这个新的例子就理应作为近似样品被给出来。如果情况不是这样,即该节点是矩形积木这一事实不具有本质意义,那么这就意味着是矩形积木这一事实为非不可缺少条件。因此,从 brick 和 wedge 两类节点开始,分别沿着 is-a 连线向上追踪,并且双方都去寻找一个共同的位于上方的节点。在现在的情况下,block(积木)就相当于这个上方节点。正是这个被支撑着的积木,作为一类节点构成了模型 4。

在这个例子中还包含着可以引伸出的内容,根据依次显示的示例,我们对生成归纳模型的启发方法进行简单地归纳和整理。

- require-link heuristic

现有模型中存在的连线在近似样品中不存在。

→其连线被置换成 must 型。

- forbid-link heuristic

现有模型中不存在的连线在近似样品中存在。

→将其连线看作是 must-not 型附加到模型上。

- climb-tree heuristic

现有模型与新的正例的对应节点属于不同类别。

→找出同处于上方的各类节点,为它们重新加上连线。

- enlarge-set heuristic

现有模型与新的正例的对应节点中,没有同类的上方节点。

→新设一个类别表示两个节点的属类别之和,并且重新为其加上连线。

- drop-link heuristic

现有模型中存在的连线在新的正例中不存在。

→其连线被清除。

最前面的两条启发方法是应用了近似样品的方法,它们一方

而使约束条件特殊化,一方面添加约束条件,从而使模型特殊化。其余各条启发方法则是基于正例的启发方法,与这些启发方法相反,放宽条件则会使模型朝着一般化方向变化。

3. 从语义网络向谓词逻辑表示的变换

语义网络、谓词逻辑、框架这样一些通用性强的知识表示,是可以进行相互变换的。

例如,在本节的说明中应用了语义网络表示,当把这种表示向谓词逻辑表示变换时,对应各连线准备2项谓词的文字,然后求它们的逻辑积就可以了。在拱的例子中,如果最后得到的是模型4那种情况,那么这时就变成

$$\begin{aligned} & \text{arch}([B, S1, S2]) \\ & \leftarrow \text{is-a}(B, \text{block}) \wedge \text{is-a}(S1, \text{brick}) \wedge \text{is-a}(S2, \\ & \quad \text{brick}) \wedge \text{support}(S1, B) \wedge \text{support}(S2, B) \wedge \\ & \quad \text{left-of}(S1, S2) \wedge \neg \text{touch}(S1, S2). \end{aligned}$$

(在学习过程中,因为要区别必然与偶然,所以 must 被用作元水平标识符,这里省略了此标识符)。如果在这里加进

$$\begin{aligned} & \text{is-a}(X, \text{block}) \leftarrow \text{is-a}(X, \text{brick}). \\ & \text{is-a}(X, \text{block}) \leftarrow \text{is-a}(X, \text{wedge}). \end{aligned}$$

这样一些积木世界的先验知识,那么拱的两个正例就可以进行说明。

上面对温斯顿的拱归纳学习进行了概括的介绍。虽然对于归纳学习来说,这是一份容易理解的好教材,但是从实际应用方面考虑,显然还存在着种种有争议的论点。例如,在负例中也好,在正例中也好,均不能给出这时的模型和只有一种性质的不同样本,因此在包含的数据等级低于复数概念的情况下,对本来的概念进行表示就会出现差距,这样便不能付诸应用,凡此等等,不再一一列举。

4.2.2 决策树的学习

逻辑式和语义网络等,在概念表示中是具有较高描述能力的知识表示方法,它们被用到了归纳学习方面,这些方法的优点是具有广泛的应用可行性,但是反过来,概念形成的处理却变得非常复杂,这时由于找不到十分有效的方法,而且搜索空间又无限扩张,计算量会变得非常庞大,因此很难实现。与此形成对照的方法,是应用决策树(decision tree)的概念表示,这种方法虽然描述能力受

到限制,但是由于计算量比较小,所以具有较高的实用价值。

作为决策树学习的著名算法,或者称为系统,被人们推荐出来的有,亨特(Hunt)等人设计的 CLS(concept learning system,1966年),昆兰的 ID3,以及作为其扩展系统的 C4.5 等。

1. 基于决策树的概念表示

对于基于决策树的学习,其事例是以特征向量,即属性-属性值对的集合来描述的,而其概念则是以决策树来表示的。所谓决策树,就是决定事例的类别的树。节点表示构成分类基准的属性,连线表示属性值,连线前端的子节点表示进一步的分类基准。在构成为叶的节点中,或被分配为正(+),或被分配为负(-)。正的节点表示,同时满足从根节点到该节点的全部分类条件的事例集合,变成了有关概念的正例,负的节点则表示相反的情况,即变成了有关概念的反例。

现在让我们用具体例子予以说明。应用下列特征:

眼睛的颜色:{黑,红}

身体的颜色:{棕,白,灰}

动物的种类:{兔、象}

设对这些特征依次进行排列,就可以表示出事例了。例如,黑眼睛的灰象就可以表示为(黑,灰,象)这种样子。这里,所谓的白化体动物概念,就可以用图 4.4 那样的决策树来表示。另外,这里不存在棕色象,并且假设在白兔和白象中也不存在白化体的东西。

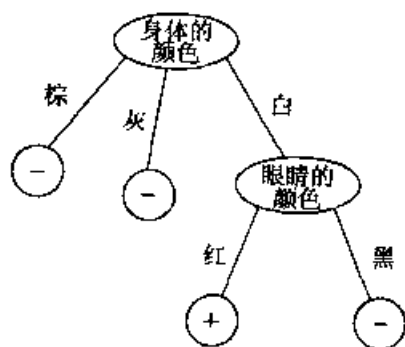


图 4.4 白化体动物的决策树

表示同一概念的决策树不限于一种。例如,就上述的白化动物的例子而言,图 4.5 的决策树与图 4.4 的决策树一样,也可以描述相同的概念。

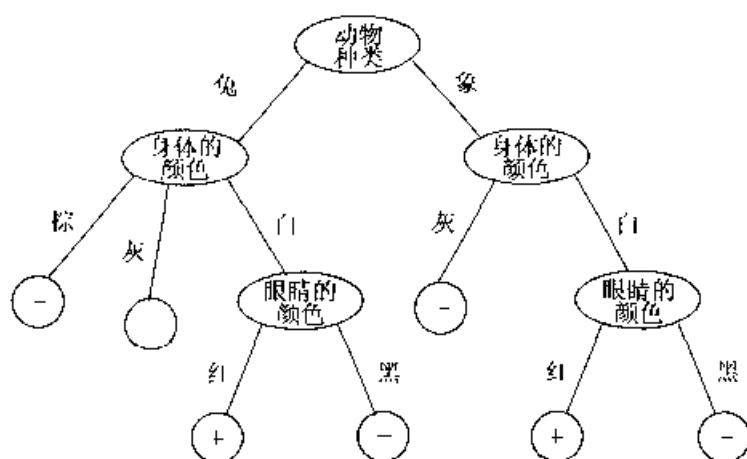


图 4.5 因为复杂而效率低的决策树

2. 决策树的学习算法

基于决策树的概念表示方法,已经是人们明白的内容,下面让我们来考虑有关决策树的学习问题。所谓决策树学习,就是根据给定的事例集合(称为训练集合),去构建用于判断事例正负的决策树。而且这种决策树不只是训练事例,而是期待它能正确地反映出事例母集合全部的性质。

如果可能有复数,则可以作为表示某个概念的决策树的候补方法,即使在这种情况下,也希望能构建更加优良的决策树。作为决策树学习算法良好程度的标准,推荐出了下列一些内容。

- **通用性** 生成的决策树是否能正确的反映母集合的全部性质。即利用这个决策树,是否也能对训练事例以外的事例进行正确分类。

- **可理解性** 其决策树的分类标准和分类步骤,是否容易为人们所理解。是否能生成与人们的思考方式相似的决策树。

- **成本** 为生成决策树所花费的成本,以及生成的决策树的应用性能。

在谈到通用性时,关于各属性值的分布和属性间的独立与从属性的假设,是必要的。例如,虽然许多的决策树学习算法采用着基于概率的评价标准,但是为使这种作法合理化,必须假定训练事例的属性值分布,与母集合全体的属性值分布达到十分完美的近似。虽然可理解性是一项困难的评价标准,但是它却是一项关系到人类的重要标准。如果人们容易对其进行理解,那么用人手去修改决策树就会变得容易,同时在参考生成的决策树并发现真正的法则过程中,会成为得力的助手。最后的成本标准可以说是一

项不言而喻的标准。在这里面,应用时的性能是一项需要先行理解,并且也是相关性很强的内容。其原因是,如果有两个具有相同分类能力的决策树,那么一般来说,越小且越简单的决策树,其性能会越高,而且同时它也容易被人们理解。

下面我们来介绍过去人们提出的一些决策树学习算法。可以说问题的关键是决策树学习算法以什么为标准选择分类属性。

- 根据经验的标准选择分类属性

决策树学习的最初系统,是亨特等人设计的 CLS。在这个系统中,最初采用的算法,是反复地进行下列各步骤的操作。

- ① 选择哪个子节点变成为正的叶节点的属性。

如果不能选择,则

- ② 同样地,选择变成为负的叶节点的属性。

如果也不能选择,则

- ③ 选择分支前某部分树中,包含的正的事例变成为最大的属性。

这是一种单纯的,同时也是与人类的直觉相一致的属性选择方法,最后得到的决策树据说会变得比较紧凑,且性能较高,但是缺乏数学根据。其后开发出的算法中,采用了基于概率论的分类属性选择方法。

- 根据信息量标准选择分类属性

当把事例的正负考虑为事件时,如果把事例集合中正负事例混杂在一起的程度与熵联系起来,并且如果确定以信息量为标准确定分类属性,那么很自然地会达到所谓好这种概念。即这是一种把信息量最多的属性(换句话说,把能使熵变成最小的属性)优先应用于试验中的方法。基于这种作法,至少对于训练来说,从根到叶节点所需要的平均试验次数可以达到概率上的最小化。这种方法构成了著名的决策学习系统昆兰 ID3,以及其发展系统 C4.5 等的基本原理。

下面对这种方法进行简单的说明。

首先,当我们把训练事例为正时的概率表示为 P^+ ,把训练事例为负时的概率表示为 P^- 时,那么关于任意训练事例正负信息的先验熵,即自身信息量 H_0 就可以表示为

$$H_0 = -P^+ \log P^+ - P^- \log P^- \quad (4.1)$$

另外,设在最初的判断中,采用某一属性 F 。这个属性的属性值集合设为 $\{v_1, v_2, \dots, v_n\}$,在训练事例中,若设属性 F 中作为 v_i

的正的事例数以 N_i^+ 表示, 同样负的事例数以 N_i^- 表示, 则当属性值作为 v_i 进行判断时, 在此条件的基础上, 与事例的正负相关的熵变成为

$$H_F = v_i = - \frac{N_i^+}{N_i^+ + N_i^-} \log \frac{N_i^+}{N_i^+ + N_i^-} - \frac{N_i^-}{N_i^+ + N_i^-} \log \frac{N_i^-}{N_i^+ + N_i^-} \quad (4.2)$$

因此, 在对属性 F 进行判断的场合用到的熵的平均值, 这时变成为

$$H_F = \sum_{i=1}^n p_i H_F = v_i = - \frac{1}{N} \sum_{i=1}^n \left\{ N_i^+ \log \frac{N_i^+}{N_i^+ + N_i^-} + N_i^- \log \frac{N_i^-}{N_i^+ + N_i^-} \right\} \quad (4.3)$$

式中, p_i 为 $F = v_i$ 的概率, N 为训练事例数。

作为用于决策树最初试验中的属性, 可以选取下列信息量的最大值:

$$\text{信息量 } \text{gain}_F = H_0 - H_F \quad (4.4)$$

因为先验熵 H_0 是通用的, 所以更换为选取能使后验熵 H_F 为最小的属性就可以了。

根据这样选择的属性, 对训练事例进行分类, 在具有某个属性值的事例全为正的事例的情况下, 在表示其属性值的连线前端应建立叶节点, 在仅有负的事例的情况下也一样。在具有某个属性值的事例集合同时包含着正和负两种事例的情况下, 应对其集合进行此前的回归处理。当然, 在上位节点已经被利用的属性, 应脱离其后补的地位。

表 4.1 事例集合举例

事例	动物种类	身体颜色	眼睛颜色	白化体
1	兔	棕	黑	负
2	兔	白	红	正
3	兔	灰	红	负
4	兔	白	红	正
5	象	白	黑	负
6	象	白	红	正
7	象	灰	红	负
8	象	灰	黑	负

让我们来看一下具体例子。设给出了表 4.1 所示的事例集合。在这个例子中, 各事例表示动物的个体。若在已知各属性值的情况下, 计算后验熵, 则得到

$$\begin{aligned}
 H_{\text{种}} &= (4/8)H_{\text{种}=\text{兔}} + (4/8)H_{\text{种}=\text{象}} \\
 &= -(1/8)\{2\log(2/4) + 2\log(2/4) + 1\log(1/4) \\
 &\quad + 3\log(3/4)\} \\
 &= 0.906
 \end{aligned}$$

$$\begin{aligned}
 H_{\text{体}} &= (1/8)H_{\text{体}=\text{棕}} + (4/8)H_{\text{体}=\text{白}} + (3/8)H_{\text{体}=\text{灰}} \\
 &= -(1/8)\{3\log(3/4) + 1\log(1/4)\} \\
 &= 0.406
 \end{aligned}$$

$$\begin{aligned}
 H_{\text{眼}} &= (3/8)H_{\text{眼}=\text{黑}} + (5/8)H_{\text{眼}=\text{红}} \\
 &= -(1/8)\{3\log(3/5) + 2\log(2/5)\} \\
 &= 0.607
 \end{aligned}$$

作为最初的分类标准,显然选择身体的颜色比较好。在身体的颜色是棕色或灰色的情况下,不论动物是什么种类,均不是白化体,在身体颜色为白色的四个例子中,正例与负例仍然混杂在一起。对于这四个例子,当把身体颜色以外的属性作为分类标准,并求其后验熵时,可以求得

$$\begin{aligned}
 H_{\text{种}} &= (2/4)H_{\text{种}=\text{兔}} + (2/4)H_{\text{种}=\text{象}} \\
 &= -(1/4)\{1\log(1/2) + 1\log(1/2)\} \\
 &= 0.5 \\
 H_{\text{眼}} &= (3/4)H_{\text{眼}=\text{红}} + (1/4)H_{\text{眼}=\text{黑}} \\
 &= 0
 \end{aligned}$$

因此,作为下一次的分类标准,选择了眼的颜色。选择了眼的颜色时的后验熵为 0,这意味着再进一步分类已经没有必要。这样得到的决策树,与图 4.4 中所示的决策树是一致的。

• 根据信息获得率选择分类属性

信息量标准在多数情况下都能很好地运作,但是当各属性的情况数中有高中趋势时,这时自然地会产生出一种论点,即在情况数多的属性中需加进偏移(即应赋予优先地位)。极端地说,如果包含着事例的 ID 那样的属性,则信息量标准必然会成为这种属性评估的首选。例如,在白化体分类的例子中,希望考虑选定事例号码作为试验属性的情况。如果事例号码被确定,则可以惟一地确定是不是白化体,因此,分类基本上可以在一次这样的试验中完成。当然这是极端的例子,如果一开始就能使 ID 那样的属性脱离对象就好了,对于情况数多的属性,不可否认的事实是具有使评估变高的倾向。例如,在图 4.6 所示的两种决策树中,右边那种决策树的评估会变高,这显然与人们的直觉相反,所以作某些正规化

处理是必要的。因此,设计出了由下式定义的信息获得率这样一种标准:

$$\text{信息获得率 } \text{gain ratio}_F = \frac{\text{gain}_F}{\text{split info}_F} \quad (4.5)$$

式中

$$\text{split info}_F = - \sum_{i=1}^n p_i \log p_i \quad (4.6)$$

这里, gain_F 是前面定义的信息量,是关于是否作为目标概念的例子的信息量。与此相对应, split info_F 是关于用于试验中的属性 F 的信息量,用这个值就把信息量进行了正规化。

例如,如果是图 4.6 所示的例子,则可以得到

$$\begin{aligned} \text{gain}_{F_1} &= (-0.075 \log 0.075 - 0.925 \log 0.925) \\ &\quad - 0.1(-0.25 \log 0.25 - 0.75 \log 0.75) \\ &= 0.384 - 0.081 \\ &= 0.303 \end{aligned}$$

$$\begin{aligned} \text{gain ratio}_{F_1} &= 0.303 / (-0.9 \log 0.9 - 0.1 \log 0.1) \\ &= 0.646 \end{aligned}$$

$$\text{gain}_{F_2} = 0.384 - 0.0 = 0.384$$

$$\begin{aligned} \text{gain ratio}_{F_2} &= 0.384 / (-0.9 \log 0.45 - 0.075 \log 0.075 \\ &\quad - 0.025 \log 0.025) \\ &= 0.265 \end{aligned}$$

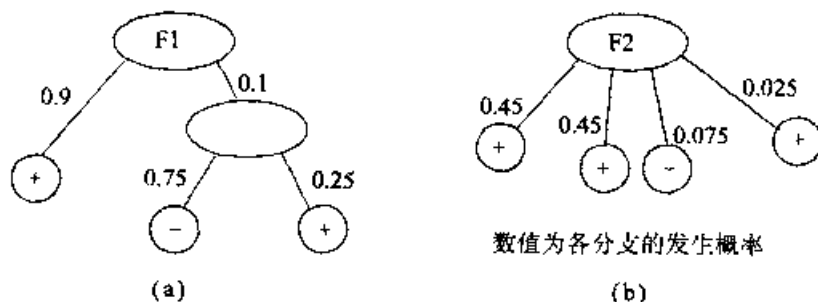


图 4.6 信息量标准与人的标准之间的差别

如果考虑以信息量作为标准,并且最初的试验选择了 F_2 ,那么相对于这种情况,如果考虑以信息量获得率作为标准时,相反地就会变成选择 F_1 。虽然不能一概而论,但是当属性的情况数多时,为获得该值付出的代价也会是高的,而且还会有使可理解性变坏的倾向。在这种情况下,采用信息获得率作为标准时,就会得到看来

似乎很有道理的决策树。

3. 从决策树向规则(谓词表示)的变换

在决策树中,从根到某个正的叶节点路径上的试验条件的逻辑积,这时变成了表示为使有关概念成立的充分条件的逻辑式。例如,对于图 4.4 中的决策树,可以生成下列规则:

$$\text{白化体}(X) \leftarrow \text{身体的颜色}(X, \text{白}) \wedge \text{眼睛的颜色}(X, \text{红}) \quad (4.7)$$

式中,设白化体(X)表示事例 X 为白化体, $F(X, V)$ 表示事例 X 的属性 F 为 V 。

同样,对于图 4.5,得到

$$\begin{aligned} \text{白化体}(X) &\leftarrow \text{动物种类}(X, \text{兔}) \wedge \text{身体颜色}(X, \text{白}) \wedge \text{眼睛颜色}(X, \text{红}) \\ \text{白化体}(X) &\leftarrow \text{动物种类}(X, \text{象}) \wedge \text{身体颜色}(X, \text{白}) \wedge \text{眼睛颜色}(X, \text{红}) \end{aligned} \quad (4.8)$$

式(4.8)中的两个式子可以归纳到一起,写成下列形式:

$$\begin{aligned} \text{白化体}(X) &\leftarrow (\text{动物种类}(X, \text{兔}) \vee \text{动物种类}(X, \text{象})) \\ &\wedge \text{身体颜色}(X, \text{白}) \wedge \text{眼睛颜色}(X, \text{红}) \end{aligned} \quad (4.9)$$

在最初的假设中,因为要从作为动物种类这种属性的兔和象两种类型值中取其一,所以可以表示为: $\text{动物种类}(X, \text{兔}) \vee \text{动物种类}(X, \text{象}) \equiv \text{True}$ 。因此,当可以消去关于动物种类的条件时,结果就得到了式(4.7)。可是,实际上还存在着其他动物种类,如果在训练的例子中碰巧只出现了两种类型的动物,则应另当别论,这时上述的简化是不成立的,因为两者的意义不同。

4. 决策树学习的发展课题

在决策树学习领域,至今仍然在进行着各种各样的发展研究工作。下面分别对它们作一些简单说明。

• 考虑信息获得成本的属性选择标准

采用信息量标准和信息获得率标准时,一般均假设,为得到属性值所付出的劳动对哪一种属性均具有均一特性,或者是与信息量成比例的。可是,现实中为获得各属性的属性值所付出的代价理应是不相同的,考虑到这种情况,不得不生成决策树。例如,如果考虑为诊断疾病的检查付出代价时,就会想像到这种情况。

• 对应于不完全训练事例的研究

在现实中,对于所有的训练事例,未必能得到全部的属性信息。事实上,多数情况下,在一部分事例中欠缺一些属性信息,得

到的属性对于每一个事例也是各不相同的。进一步说,给出的训练事例也未必只是通常所说的正确的事例(注意,它和正的事例的意义是不同的)。针对包含有噪声即错误事例的情况,其处理方法也是当前的研究课题。

- 允许分类误差与决策树剪枝

在实际问题中,为了达到100%的分类率,学习自身的代价和分类的代价都会变得很大而脱离实际。另外,如果在训练的事例中存在着混入噪声的可能性,那么100%的分类原本就是不可能的。因此,索性允许某种程度的分类误差,然后在同等程度的误差率时,去寻求一种利用尽量少的训练事例,构成更简单的决策树的算法。在训练事例不充分的情况下,能对给出针对全体集合误差率的训练事例进行说明的决策树,未必比相反情况的决策树差,这是很清楚的。这种现象是一种过匹配(over fitting),为防止产生这种情况的剪枝方法等,正处于研究过程中。

4.3 根据丰富的知识和经验提高推理效率

在4.2节中,根据介绍的例子得到的归纳学习几乎没有假设先验知识,把多数事例的类似性和统计性质作为线索来生成新的知识。与此相对应,本节中要介绍的效率化学习则假设了丰富的先验知识。对于少数的事例(问题),可以把运用其先验知识得到的实际推理结果作为基础,进而获得必要的知识。

4.3.1 效率化学习

一般来说,解决问题可以归结为树或图的路径搜索问题。如果搜索空间是静止而不发生变化的,那么对于同样问题能够获得的解,基本上应该是相同的,过去用试探的方法进行广泛地搜索未能得到解的选择分支,即使再尝试着进行几次搜索仍然不可能求得解答。即使是在对一个问题的求解过程中,相同部分的问题反复出现的情况也是常有的。在这种情况下,对于能解决的部分问题,如果对其解答进行存储,并且再次利用,那么就可以避免进行无效地重复性劳动。这种考虑方法,在使解决问题效率化的过程中,自然会成为一种有效的方案,因此在多数的问题求解系统中,被用来作为加速搜索过程的一种技术手段。这种技术手段,使得

人们能够在初次经验的基础上进行搜索,从而在第二次及其以后的搜索过程中,得以比初次更好(更高速)地解决问题,正是在这种意义上,才称其为效率化学习(speedup learning)。

这种方法不只是作为一种技术,而且它还作为系统的学习机能而得到了初步研究,在这方面,著名的 STRIPS (Stanford, research institute problem solver) 的宏操作器 (macro-operator), 作为机器人的行动计划系统被推荐了出来。另外,具有同样的功能,但对广泛的问题进行求解的系统 Soar, 被称为冠军王 (chanking)。宏操作器也好,冠军王也好,都是把产生式系统的子目标与其实施结果组合在一起,生成新的产生式规则的技术。当利用这样构成的规则时,就可以省略中途的演绎过程(产生式系统的认识—行动循环),一跃而直接达到结论,从而可以加快推理。在专家系统领域,这样的功能被称为源于深奥知识的浅显知识的知识编译。

设对 STRIPS 以后被研究的效率化学习,用逻辑的观点进行统一的讨论,框架就成了基于解释的学习 (EBL)。如果以逻辑的观点看效率化学习,那么这就相当于下列事实,即“当 $A \rightarrow B$ 和 $B \rightarrow C$ 这种逻辑式被包含在先验知识中时,如果 ' $A \rightarrow B$ 且 $B \rightarrow C$ ', 则有 ' $A \rightarrow C$ ' 这种推理在证明过程中频繁出现,那么 ' $A \rightarrow C$ ' 也会包含在先验知识中”。在这个过程中采用的推理,无非是演绎推理。因此,效率化学习也称为演绎学习。另外,像 4.2 节中介绍的拱学习那样,基于事例的类似性的学习,与 EBL 相对应,称之为 SBL (similarity based learning, 基于类似性的学习)。

下面对 EBL 进行简单说明。

4.3.2 基于解释的学习 (EBL)

考虑作为知识表示的谓词逻辑式。一般来说,在用逻辑表示的知识处理系统中,是在定理的证明中,将解与证明树对应起来考虑问题的求解的。在定理证明中,虽然在最终得到一种证明树之前会有非常多的试探性的错误,但是如果前提是相同的,则相同子目标(部分逻辑式)的证明结果,通常应该变成相同的。因此,同样的子目标在第二次出现时,会再次发生同样的试探性错误,因而造成明显的浪费。把证明完了的子目标的证明树进行存储,如能再度利用这种存储结果,那么就可以避免这种浪费,从而可以提高推理效率。这种用来作为学习的考虑方法,乃是 EBL 方法。

EBL 的输入和输出有下列一些种类。

【输入】

目标概念 要学习的概念

训练事例 目标概念的一个事例

领域知识(domain knowledge) 在对象领域内成立的知识

操作性准则(operationality criterion) 作为学习结果得到的目标概念的定义规则应满足的条件

【输出】

满足操作性准则的关于目标概念的充分条件。

EBL 的执行步骤由下列两步组成:

第 1 步(解释)

根据领域知识,证明训练事例是目标概念的一个例子。这个证明即为“解释”。

第 2 步(一般化)

把上面得到的证明树中的常数变量化,据此可以实现一般化。从作为结果而得到的树,选择满足操作性准则的那样的规则的记述,从而构造出目标概念的定义规则。

下面我们举一个具体例子。

目标概念

前进可能(X)

训练事例

色(carA,红). 色(carB,黑). 色(信号,蓝). 发动机(carA,on). 发动机(carB,on). 齿轮(carA,d). 齿轮(carB,p). 制动器(carA,off). 制动器(carB,on). 前方障碍(carA,none). 前方障碍(carB,carA). 车灯(carA,on). 车灯(carB,off). daytime.

领域知识

前进可能(X) \leftarrow 前方安全(X) \wedge 行驶可能(X).

前方安全(X) \leftarrow 色(信号,蓝) \wedge 前方障碍(X,none) \wedge 视野良好(X).

行驶可能(X) \leftarrow 发动机(X,on) \wedge 齿轮(X,d) \wedge 制动器(X,off).

视野良好(X) \leftarrow daytime \vee (nighttime \wedge 车灯(X,on)).

操作性准则

概念记述就是用训练事例中的谓词进行记述。

以上述输入作为基础,进行前进可能(X)的证明,可以得到图 4.7(a)表示的证明树。对该树进行一般化处理,得到图 4.7(b)所

示的一般化证明树。根据这个结果,作为满足操作性规范的概念定义,可以表示如下:

前进可能(X) \leftarrow 色(信号, 蓝) \wedge 前方障碍(X , none) \wedge daytime
 \wedge 发动机(X , on) \wedge 制动器(X , off) \wedge 齿轮(X , d).

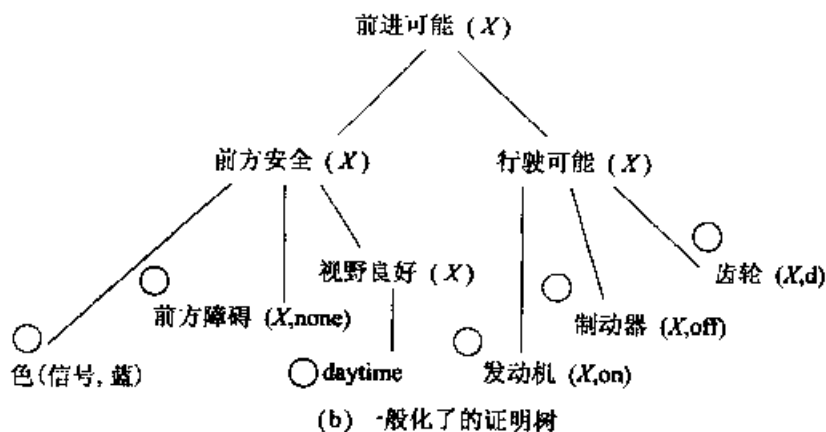
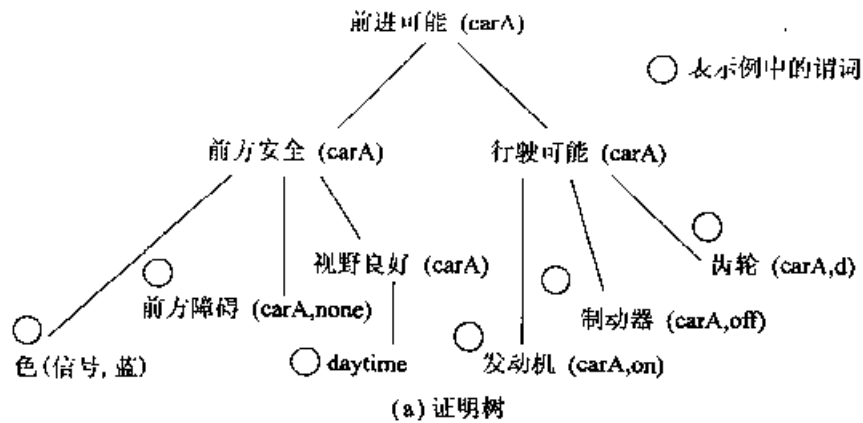


图 4.7 前进可能概念的说明

实际上,用 EBL 得到的概念定义规则,即使没有训练事例也可以生成。用 EBL 生成的规则,不过是根据先验知识演绎推理得出的逻辑结果。但是如果说事例是没有意义的,也并非属实。随随便便由逻辑式集合演绎推理可以导出的规则,用来解决实际问题是没有根据的,它只能造成存储容量和搜索时间方面的浪费。为了能从演绎可能导出的规则中,提取出在解决实际问题时可以应用的部分,事例就是有意义的了。

下面让我们来归纳 EBL 的特征。EBL 的优点如下:

(1) 得到的知识由于是通过演绎得到的,所以是健全的知识,

即是正确的知识。与此形成对照, SBL 也许会造成知识的纯粹增加, 这会伴随着产生错误知识的危险性。

(2) 不需要很多的事例, 只要根据少数事例就可以得到有效的规则。根据丰富的先验知识和少数的事例就可以获得知识, 这种状况是非常现实的, 它接近于我们人类的现实状况。

(3) 作为 EBL 基础的演绎推理机构, 可以与问题求解系统共有。

(4) 问题求解系统本身, 在开发和应用方面取得了许多实际成绩, 由于它可实现高效率, 所以在这个意义上它是一种有实用价值的学习。凡此等等, 均为 EBL 的优点。

反之, EBL 作为效率化学习的初期方法, 它有下列一般性缺点:

(1) 除了问题求解时间缩短这一点外, 这种方法不能提高对新问题的解决能力, 在纯粹知识的增加方面, 也不会作出贡献。因此, 也有一种见解, 认为效率化学习只是一种技术, 在狭义上不认为它是一种学习。

(2) 效率化学习实际上与效率化无关、有时甚至会发生使效率降低的情况。通过效率化学习得到的规则, 可以说是冗长的规则, 求解问题时的规则(知识)的搜索空间广阔。因为搜索空间的增大一般与搜索时间的增加有关, 所以在利用效率化规则时, 搜索空间增大与导致时间缩短之间的折衷关系便随之而产生了。在这种情况下, 采用由什么样的标准进行学习而生成的规则, 就成了要考虑的问题。这种问题称为**效用问题**(utility problem)。

最后, 我们对 SBL 与 EBL 进行比较, 并将比较结果汇集于表 4.2 中。

表 4.2 SBL 与 EBL 的比较

	SBL	EBL
训练事例	需要许多事例	只需一个或少数事例
先验知识	需要的先验知识少, 需要知道定义域和类别关系的知识水平	需要丰富的先验知识和足以能够说明事例的有关内容
推理的种类	归纳	演绎
生成知识的健全性(正确性)	没有保证	有保证
本质知识的增加	有	无
学习的目的	形成概念	高效化

媒体(agent)

具有自己的行动规范和内部状态并能与外界进行交流的独立行动体,称为**媒体**。媒体通过**传感器(sensor)**从外界获得信息。以这种信息和内部状态作为基础,可以认识环境和环境自身的处境,然后再以自己的行动规范为基础,决定自己应该采取的行动。媒体具有对环境产生影响的**效果器(effector)**,效果器是根据预先的决定,对外界产生作用的。

例如,人类和智能机器人就是具有代表性的媒体。人类这种媒体的传感器是眼睛、耳朵和触觉,效果器是手、足和口。如果把人和机器人等作为媒体的例子,可能会造成误解即认为媒体必须具有眼睛能够看到的形体。实际不是这样,计算机处理等中的软件也可以作为媒体。同样,被称为外界或环境的东西,也未必是实际的周围世界,由计算机内部或者因特网形成的抽象空间也可以看作是外界环境。

在存在多数媒体的情况中,媒体被称为**多媒体(multi-agent)**。在媒体这种概念中,媒体间的直接或间接通信,是它的一种附带功能。如果只有一种媒体,或者即使有多种媒体,但是互相之间都不施加任何影响,那么就没有必要称它们为媒体。某个行动体如果是完全独立于其他媒体,那么它只可能是自立的。因此媒体系统通常情况下都会变成多媒体系统。

另外,由于必须构成多媒体,所以媒体这种东西显然不是全知全能的。如果是全知全能的,通信就没有必要了。一种媒体只具有局部的信息,或者说只具有解决部分问题的功能,这是理所当然的。因此便产生了与其他媒体的通信,而且作为媒体的全体,有可能去处理非常复杂的问题。

在这一章中,我们把机器学习这一项目,以媒体的形式进行了研究。教师也是一种媒体,它具有对学生进行教育的意图,并且与学生之间进行通信联系。另外,还可能有关于共同作业的学习方面的问题出现。在这种情况下,学习的媒体会变成多数媒体而存在于机器学习中。

练习题

1. 试分别举出死记学习、归纳学习、效率化学习和强化学习方面的例子。
2. 在由例子进行归纳推理方面,试针对采用谓词逻辑表示和采用决策树表示两种情况,分别概括出它们各自的优缺点。
3. 试概括 SBL 和 EBL 的特征,对两种方法进行比较,并讨论它们各自的优缺点。
4. 试以 EBL 为代表,在效率化方面,研究造成效用问题的原因及其解决方法。

5. 当给出下列输入时,试说明基于 EBL 学习的情况:

目标概念:

杯子(X)

训练事例:

把手样式(cupA), 色(cupA , 白), 向上呈凹型(cupA), 重量(cupA , 0.5), 平底(cupA).

领域知识:

杯子(X) \leftarrow 稳定(X) \wedge 能举起(X) \wedge 保存液体(X).

稳定(X) \leftarrow 平底(X).

能举起(X) \leftarrow 轻的(X) \wedge (把手样式(X) \vee 可以把握(X)).

轻的(X) \leftarrow 重量(X, W) $\wedge W < 2$.

保存液体(X) \leftarrow 向上呈凹形(X).

操作性准则:

训练事例中的谓词以内部谓词形式定义目标概念。

第 5 章

模糊理论-神经网络-遗传算法

一般认为人类是逻辑思考与直觉思考并用。在过去的人工智能研究中,对人类的逻辑思考给予了特别关注,并且构建了算法,基于符号处理的推理和知识框架。

另一方面,阐明人类直觉思考的尝试始于 1960 年前后。对于以前的人工智能,处理人类的含糊性和灵活性的是模糊理论,模拟人类脑神经系统学习机能的是神经网络,对生物的生命进化进行模型化的是遗传算法。这些内容,作为人工智能的新方法,受到了人们的重视。近年来,与这些方法相辅相成,利用了同时融合的方法,为了达到易使用、鲁棒性和低成本,不一定要求高度的精确性和可靠性,正在进行探讨和研究采用低精度计算的不精确性和不可靠性的容许程度问题。这种新的计算模式称为软计算(soft computing)。本章在软计算中,对模糊理论,神经网络和遗传算法,给予了特别关注,下面将对相关内容进行概括的介绍。

5.1 模糊理论

5.1.1 什么是模糊理论

模糊理论^[1]是 1965 年由扎德(L. A. Zadeh)提出来的。所谓模糊(fuzzy),可以解释为诸如“含糊”、“不明确”之类的意思。扎德注意到了人类直觉的正确判断能力和行动能力,为了能把人类的含糊性作为数学模型进行处理,模糊理论便应运而生。

在模糊理论中,通常是集合的境界变得含糊不清,因而采用了扩张的模糊集合。模糊理论是以控制领域、计算机领域及 OR(operations research)领域为中心进行研究的^[2~4]。特别是模糊控

制^[1,2]是一种有效的方法。模糊控制是把控制经验方面的知识,记述在包含模糊集合的 if-then 规则中,并且用推理处理得出结果的一种方法。由扎德提出的模糊理论,1974 年以马姆丹尼(E. H. Mamdani)开发的蒸汽机自动控制为契机,开始变成为实用化技术。1980 年,丹麦的 F. L. Smith 公司开发了水泥窑用的模糊控制器,1987 年,由日立制作所研制的模糊控制器,实现了对日本仙台市地下铁道的自动运行控制,凡此等等,都对模糊理论的实用化起到了促进作用。1989 年,在松下电器工业部门制造的浴缸系统中,由于应用模糊控制而开发了恒温控制装置,1990~1991 年间,以家电领域为中心,开发出了许多模糊控制产品。这就形成了所谓的“模糊风暴”。据称现在已经有了大约 500~1 000 个应用实例。

5.1.2 模糊集合与普通集合的区别

现在我们来考虑“青年”这个集合,这个集合用 18 岁以上,25 岁以下来定义(图 5.1)。可是,对于 17 岁 364 天也不能说“不是青年”。因此,这里的“青年”和“不是青年”的境界是含糊不清的。在模糊集合中,对于年龄的年轻程度,用 0~1 的值给出。例如,24 岁的青年其年轻程度可以认为是 0.8 等。

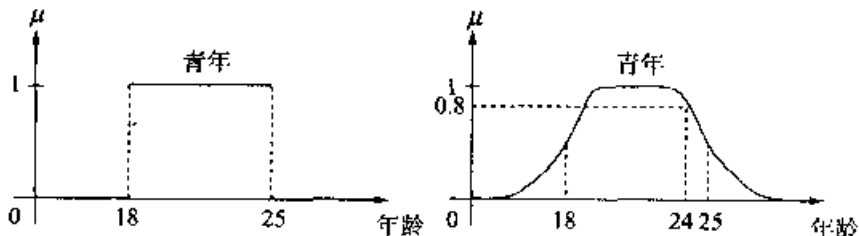


图 5.1 普通集合与模糊集合

具有某种属性的事物的全体集合 X ,其模糊集合 A ,是一种具有由下列隶属函数 μ_A 决定的特性的集合:

$$\mu_A: X \rightarrow [0,1] \quad (5.1)$$

式中, $\mu_A(x) \in [0,1]$,它表示对 X 的元素 x 的模糊集合 A 的隶属程度,称为隶属值。例如在图 5.1 中,模糊集合 A 表示“青年”, $\mu_{\text{青年}}(24) = 0.8$ 。

模糊集合也可以在 X 的元素 x 为有限离散量的情况下定义。现在让我们来考虑女性全体的集合 $X = \{\text{理江, 有利沙, 并惠, 静夏, 广江, 诚子}\}$ 。一位男性朋友根据自己的主观判断,对这些女性

的“身材矮小”程度作出了下列判断：

$$\mu_A(\text{理江})=0.5, \mu_A(\text{有利沙})=0.0, \mu_A(\text{并惠})=1.0$$

$$\mu_A(\text{静夏})=0.3, \mu_A(\text{广江})=0.8, \mu_A(\text{诚子})=0.7$$

根据这些隶属值,可以在 X 上对“身材矮小女性”的模糊集合 A 进行定义。模糊集合 A 可以表示如下(图 5.2)：

$$A = 0.5/\text{理江} + 0.0/\text{有利沙} + 1.0/\text{并惠} + 0.3/\text{静夏} \\ + 0.8/\text{广江} + 0.7/\text{诚子} \quad (5.2)$$

式中, / 为分离符号, + 表示“或(OR)”。 $\mu_A(x)/x$ 表示 x 对 A 的隶属值是 $\mu_A(x)$ 。但是,也存在除掉 $\mu_A=0$ 的元素的情况。

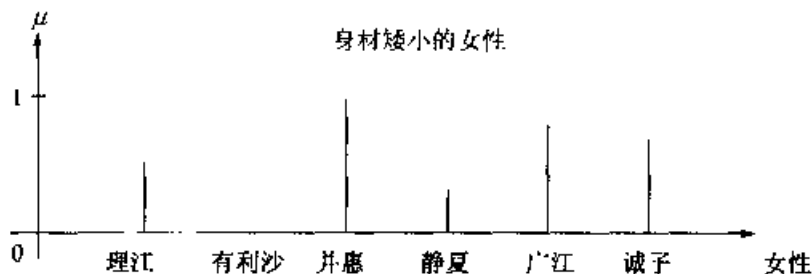


图 5.2 身材矮小女性的模糊集合

另一方面,普通集合的值是 0 或 1 这两个值。因为隶属值的领域 $\{0,1\}$ 被包含在了闭区间 $[0,1]$, 所以普通集合是模糊集合的特殊情况。相对于模糊集合,普通集合被称为轮廓分明的集合。在轮廓分明的集合中,定义了种种集合运算法则,与此相同,在模糊集合中也可以定义相应的集合运算法则。模糊集合的运算可以定义如下：

$$\text{相等: } A=B \Leftrightarrow \mu_A(x)=\mu_B(x) \quad (5.3)$$

$$\text{包含: } A \subset B \Leftrightarrow \mu_A(x) \leq \mu_B(x) \quad (5.4)$$

$$\text{补集: } \bar{A} \Leftrightarrow \mu_{\bar{A}}(x)=1-\mu_A(x) \quad (5.5)$$

$$\text{并集: } A \cup B \Leftrightarrow \mu_{A \cup B}(x)=\mu_A(x) \vee \mu_B(x) \quad (5.6)$$

$$\text{交集: } A \cap B \Leftrightarrow \mu_{A \cap B}(x)=\mu_A(x) \wedge \mu_B(x) \quad (5.7)$$

式中, \vee 表示 \max 运算, \wedge 表示 \min 运算。在图 5.3 上,表示了模糊集合 A 与 B 的并集 $A \cup B$ 及交集 $A \cap B$ 的例子, A 的补集 \bar{A} 表示在图 5.4 中。

当模糊集合作为离散集合时,让我们来列举一些模糊集合的运算例子。例如,设 $X=\{\text{雅弘, 强, 卓也, 五郎, 伸吾}\}$, 现在对 A —“年轻的”, B —“身材高的”定义如下：

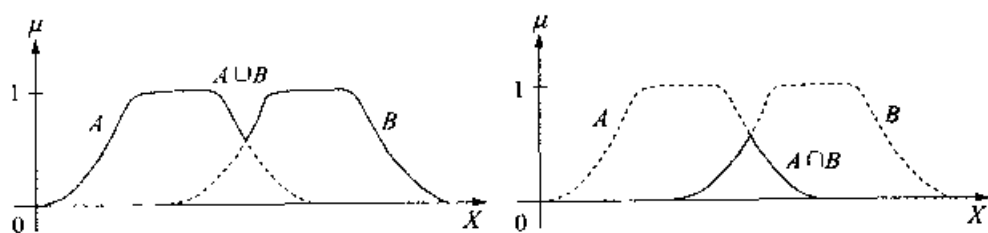


图 5.3 模糊集合的并集合与交集合

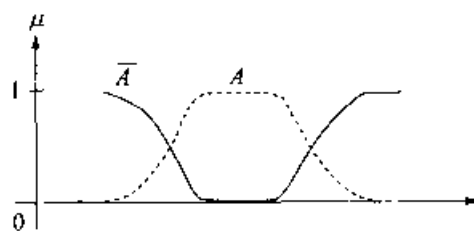


图 5.4 模糊集合的补集合

$$A = 0.4/\text{雅弘} + 0.6/\text{强} + 0.8/\text{卓也} + 1.0/\text{五郎} \\ + 0.9/\text{伸吾} \quad (5.8)$$

$$B = 0.3/\text{雅弘} + 0.5/\text{强} + 0.9/\text{卓也} + 0.6/\text{五郎} \\ + 1.0/\text{伸吾} \quad (5.9)$$

并集 $A \cup B$ 表示“年轻的和身材高的”构成的模糊集合,交集 $A \cap B$ 表示“年轻且身材高的”构成的模糊集合,补集 \bar{A} 表示“不年轻的”构成的模糊集合,这些模糊可以表示如下:

$$A \cup B = 0.4/\text{雅弘} + 0.6/\text{强} + 0.9/\text{卓也} + 1.0/\text{五郎} \\ + 1.0/\text{伸吾} \quad (5.10)$$

$$A \cap B = 0.3/\text{雅弘} + 0.5/\text{强} + 0.8/\text{卓也} + 0.6/\text{五郎} \\ + 0.9/\text{伸吾} \quad (5.11)$$

$$\bar{A} = 0.6/\text{雅弘} + 0.4/\text{强} + 0.2/\text{卓也} + 0.0/\text{五郎} \\ + 0.1/\text{伸吾} \quad (5.12)$$

对于模糊集合,德莫尔根(De-Morgan)法则成立¹⁾。式中,对于全集 X 的任意模糊集合 A ,存在 $A \cap \bar{A} \neq \emptyset$, $A \cup \bar{A} \neq X$ 。

5.1.3 模糊数也是数吗?

在日常生活中,例如经常说“大约1000元”,“到机场需要90

1) $\overline{A \cap B} = \bar{A} \cup \bar{B}$, $\overline{A \cup B} = \bar{A} \cap \bar{B}$

分钟左右”等等,像这样采用含糊数字的情况是很多的。这样一些数是可以用模糊集合表示的。杜波依斯(D. Dubois)和普雷德(H. Prade)把满足下列三条性质的模糊集合称为模糊数。

(1) 连续性: $E_\lambda = \{x | \mu_A(x) \geq \lambda\}$ 是闭区间。 (5.13)

(2) 规范性: $\mu_A(x) = 1$ 时的 x 存在。 (5.14)

(3) 凸性: $\mu_A(\lambda x_1 + (1-\lambda)x_2) \geq \mu_A(x_1) \wedge \mu_A(x_2), x_1, x_2 \in X$ (5.15)

式中, $0 \leq \lambda \leq 1$ 。一般情况下,模糊数是以数直线上模糊集合出现的。

因为模糊数是数的扩张,所以与普通数的情况相同,对模糊数也可以定义加减乘除。模糊数的运算可以用下列扩张原理予以定义。

【扩张原理】

对于数直线上的数 x_1, x_2 , 当扩张两项的运算 $x_1 * x_2$ 为两项模糊数 A, B 的运算 $A * B$ 时,可以定义如下:

$$C = A * B: \mu_C(y) = \sup_{x_1 * x_2 = y} \{\mu_A(x_1) \wedge \mu_B(x_2)\} \quad (5.16)$$

现在我们设 $\tilde{2}$ 和 $\tilde{4}$ 分别为“大体上的 2”和“大体上的 4”这样两个模糊数。根据扩张原理, $2+4$ 的运算可以扩张为 $\tilde{2} + \tilde{4} = \tilde{6}$ 的模糊数的运算(图 5.5)。

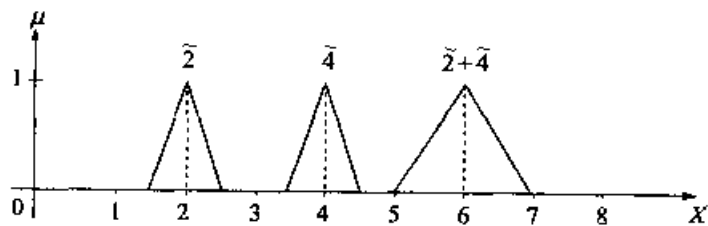


图 5.5 模糊数 $\tilde{2} + \tilde{4}$ 的运算

在扩张原理中,当模糊数的元素增多时,计算会变得复杂化。根据 α 割集¹⁾,采用区间解析法,可以简化模糊数的运算。

现在设模糊数 M, N 的 α 割集为

$$M_\alpha = [a, b], N_\alpha = [c, d] \quad (5.17)$$

于是区间 M_α, N_α 的运算变成为下列形式:

1) $A_\alpha = \{x | \mu_A(x) \geq \alpha\}, 0 \leq \alpha \leq 1$

$$M_a + N_a = [a + c, b + d] = (M + N)_a \quad (5.18)$$

$$M_a - N_a = [a - d, b - c] = (M - N)_a \quad (5.19)$$

$$M_a \times N_a = [a \times c, b \times d] = (M \times N)_a \quad (5.20)$$

$$M_a \div N_a = [a \div d, b \div c] = (M \div N)_a \quad (5.21)$$

基于区间解析法的区间,与扩张原理的结果是一致的。

5.1.4 模糊控制是一种方便的控制方法

在这里,我们以引入了模糊控制的洗衣机为例^[5]进行说明,并且介绍有关模糊控制的推理方法。作为该例子,要根据被洗涤衣物的污染性质和数量,在不损伤衣物并清洗掉所有污垢的情况下,决定洗涤时间。在模糊控制中,将采用洗涤高手的经验和窍门,并且把这种知识以 if then 形式的规则表示出来,然后根据模糊控制,确定出最佳洗涤时间。

图 5.6 示出了洗衣机的剖面图。在排水阀附近的光电传感器部件中,光电晶体管会感受到来自发光 LED 发出的光强的作用,待变换成电压后,可以计算出洗涤水的污染情况。洗涤水的透射率随时间的变化关系如图 5.7 所示。在开始洗涤时,衣服上的污垢被溶解到洗涤水中,于是透射率下降。这时,可以探测出透射率的下降速度。在灰尘污染衣物的情况下,污垢只是容易使洗衣机的机械搅拌力迅速下降,在衣物受到油脂污染的情况下,洗涤剂的洗涤能力会减弱,并且会被溶解掉(图 5.7(a))。因此,利用达到污染饱和点时的饱和时间(下降速度)可以检测出是灰尘污染还是

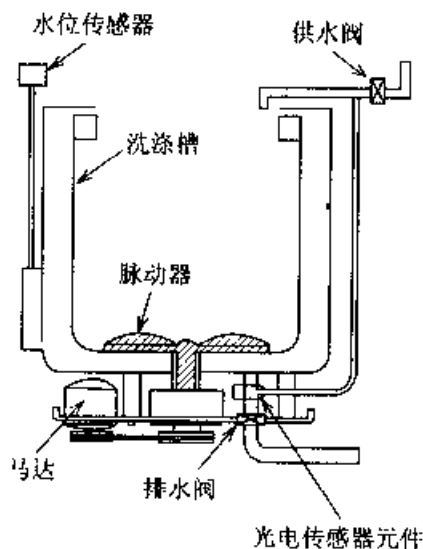


图 5.6 洗衣机的剖面图

油脂污染。另一方面,污染程度,可以通过达到饱和时的透射率得到(图 5.7(b))。在透射率低的情况下,污染程度比较严重,透射率高的情况下,污染程度则比较轻。

在模糊控制中,采用了 if-then 型的产生式规则。因为在规则中应用了模糊集合,所以称这种规则为模糊规则。在洗衣机这个例子中,首先,以在一般家庭中监测到的数据和洗涤软件开发者的知识作为基础,制订出有关油脂污染和严重污染的规则。然后,由于这些规则表示了增加洗涤时间的信息,所以可以把它们作为一个规则进行整理,并且变换成饱和时间和透射率规则。最后,作为模糊规则,对它们进行描述。这样,在总体上制订六个模糊规则。

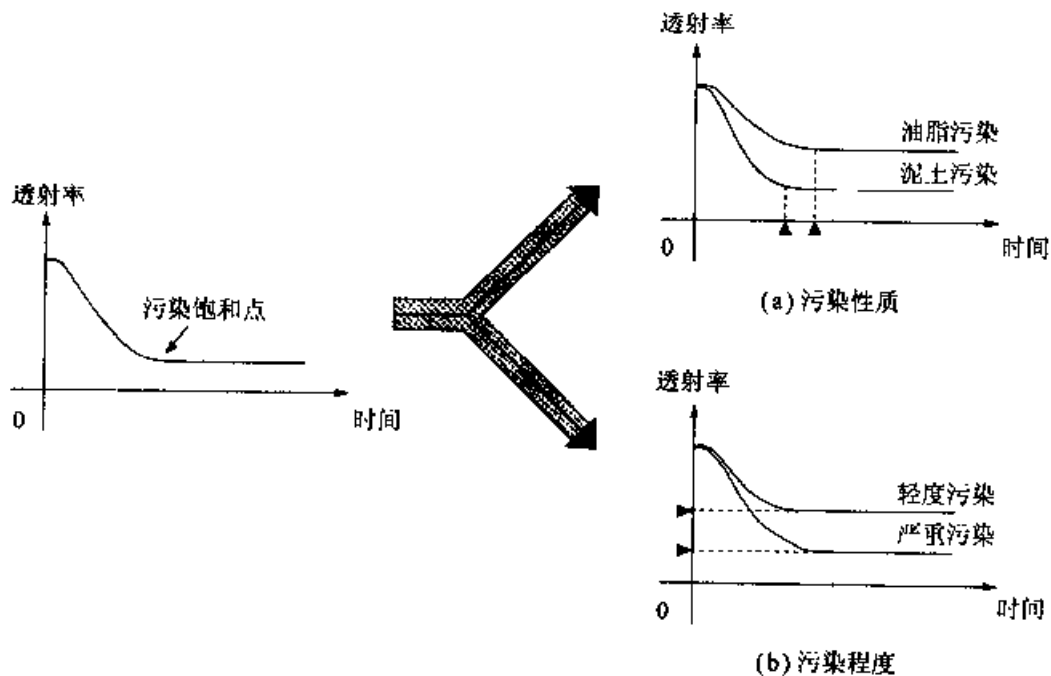


图 5.7 洗涤水的透射率随时间的变化

• 一般家庭的数据和洗涤软件开发者的经验:

“油脂污染比灰尘污染的洗涤时间长”

“严重污染时洗涤时间长”

↓ 归纳成规则

• 根据衣物的污染性质和污染程度得出的规则:

“if 是比灰尘污染更甚的油脂污染,并且污染情况严重,then 洗涤时间应大大加长”

↓ 把规则变换成饱和时间和透射率

• 由饱和时间和透射率得到的规则:

“if 饱和时间长, 并且设透射率低, then 洗涤时间应大大加长”

↓ 变换成模糊规则

• 用于模糊控制的第一模糊规则:

“if T is Big and V is Small then t is Very Big”

式中, T 为饱和时间, V 为表示透射率的变量, t 是表示洗涤时间的变量。另外, Big、Small、Very Big 是表示模糊数的一种水平。称这种模糊数为模糊水平。

此外, 作为第二项模糊规则, 考虑下列规则:

“if T is Big and V is Medium then t is Big” (5.22)

一般来说, 模糊规则可以设为具有下列形式的 r 个规则 $R_s, s = 1, 2, \dots, r$ 。

$$R_s: \text{if } x_1 \text{ is } F_{s1} \text{ and } \dots \text{ and } x_n \text{ is } F_{sn} \text{ then } y \text{ is } B_s \quad (5.23)$$

式中, B_s 和 F_{sj} 为模糊水平, 且 $s = 1, 2, \dots, r, j = 1, 2, \dots, n$ 。称 if 部分为模糊规则的事件前部, then 部分为模糊规则的事件后部。

在模糊控制中, 虽然提出了种种推理方法^[1], 但是麦姆德尼的推理方法仍不失为一般的推理方法。现在设已知输入数据为 $x_1^*, x_2^*, \dots, x_n^*$ 。结果的模糊集合 B^* 可以求得如下:

$$\begin{aligned} \mu_{B^*}(y) = & ((\mu_{F_{11}}(x_1^*) \wedge \mu_{F_{12}}(x_2^*) \wedge \dots \wedge \mu_{F_{1n}}(x_n^*)) \wedge \mu_{B_1}(y)) \vee \dots \vee \\ & ((\mu_{F_{r1}}(x_1^*) \wedge \mu_{F_{r2}}(x_2^*) \wedge \dots \wedge \mu_{F_{rn}}(x_n^*)) \wedge \mu_{B_r}(y)) \end{aligned} \quad (5.24)$$

在控制的情况下, 结果与其以模糊集合的形式求得, 不如以实数的形式求得更为方便。一般情况下, 要计算结果的模糊集合重心, 设重心为实数 b^* 。于是

$$b^* = \frac{\int_Y y \times \mu_{B^*}(y) dy}{\int_Y \mu_{B^*}(y) dy} \quad (5.25)$$

作为其他推理方法, 还提出了 \max - \times 运算等推理方法, 以取代基于 \wedge 和 \vee 的 \max - \min 运算。另外, 在式(5.25)中, 虽然可以计算模糊集合 B^* 的重心值 b^* , 但是也还存在着采用模糊数的中央值和最大值等的方法。式(5.25)的变换称为非模糊化。

为了具体说明问题, 采用前面的关于洗涤时间的例子, 对模糊控制的推理方法进行说明。这时, 作为输入数据, 设已知饱和时间 $T=10\text{min}$ 和透射率 $V=30\%$ 。则用第一规则的结果的模糊数, 可以根据下式求得:

$$\mu_{B_1}(t) = (\mu_{Big}(10) \wedge \mu_{Small}(30)) \wedge \mu_{Very\ Big}(t) \quad (5.26)$$

图 5.8 中的斜线部分表示了结果的模糊数 $\mu_{B_1}(t)$ 。同样,关于第二规则的结果的模糊数 $\mu_{B_2}(t)$ 也可以相应地求得。作为模糊控制的结果,求取第一规则和第二规则的 max 运算的模糊集合重心,可以求得为 $t=13\text{min}$ 。

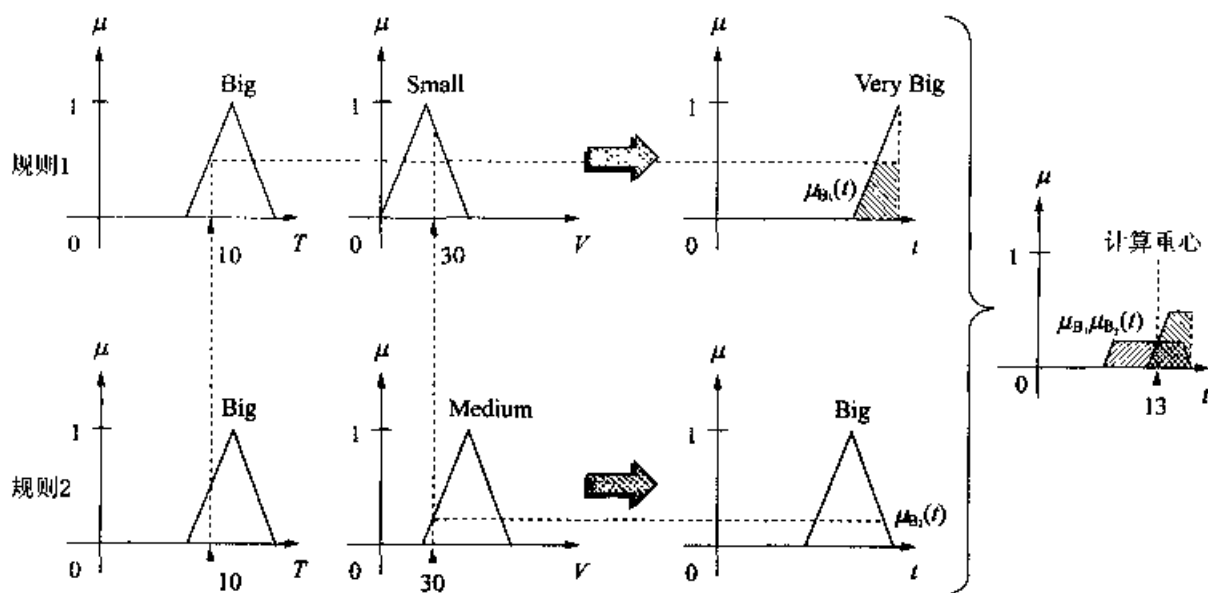


图 5.8 模糊控制的推理过程

最后,让我们来介绍菅野的模糊控制,在这种控制中,把规则的事件后部设成了线性方程式。在菅野的模糊控制中,对第 s 号模糊规则定义如下:

R_s : if x_1 is F_{s1} and \cdots and x_n is F_{sn}

then $y_s = w_{s0} + w_{s1}x_1 + w_{s2}x_2 + \cdots + w_{sn}x_n \quad (5.27)$

式中, x_1, x_2, \cdots, x_n 为线性方程式的输入变量, y_s 为输出变量, $w_{s0}, w_{s1}, \cdots, w_{sn}$ 为系数。

已知输入数据 $x_1^*, x_2^*, \cdots, x_n^*$ 时,结果 b^* 可以根据下式求得:

$$y_s^* = w_{s0} + w_{s1}x_1^* + w_{s2}x_2^* + \cdots + w_{sn}x_n^* \quad s=1, 2, \cdots, r \quad (5.28)$$

$$g_s = \mu_{F_{s1}}(x_1^*) \times \mu_{F_{s2}}(x_2^*) \times \cdots \times \mu_{F_{sn}}(x_n^*) \quad s=1, 2, \cdots, r \quad (5.29)$$

$$b^* = \frac{\sum_{i=1}^l g_i \times y_i^*}{\sum_{i=1}^l g_i} \quad (5.30)$$

因为在实际的模糊控制中,采用了线性方程式,所以可以有效地应用到处理控制方程式的控制问题中,以及有效地应用到利用判别方程式的模式问题等情况中。

模糊控制的结果,受模糊规则数和隶属函数形状的影响。因此,为了能得到期望的结果,有必要对模糊规则和隶属函数进行调整。这种调整工作称为调准。在5.2节中,将具体说明一个利用神经网络的调准方法进行调准的例子。

模糊控制和模糊推理

模糊控制是模糊推理的应用。模糊推理是二值逻辑推理的模糊化。现在让我们考虑 if x is A then y is B 这个模糊规则。

在模糊推理中,根据 x is $A \rightarrow y$ is B (简略的记作 $A \rightarrow B$) 蕴含的运算,对 x is A^* 进行合成运算,即可得到 y is B^* 的结果。这种模糊推理可以表示如下:

$$B^* = A^* \circ (A \rightarrow B)$$

式中, \circ 表示 max-min 等的合成运算。

另一方面,在模糊控制中,结果 B^* 可以由下式得到:

$$B^* = (A^* \circ A) \rightarrow B$$

因此,这些结果是不同的。可是,在蕴含运算与合成运算的第一个运算相一致的情况下,结果会变成相同的。例如,设蕴含运算为 min 运算,并且设合成运算为 max-min 运算时,模糊推理的结果 B^* 的隶属函数 $\mu_{B^*}(y)$ 变为

$$\begin{aligned} \mu_{B^*}(y) &= \max_x (\mu_{A^*}(x) \wedge (\mu_A(x) \wedge \mu_B(y))) \\ &= \max_x ((\mu_{A^*}(x) \wedge \mu_A(x)) \wedge \mu_B(y)) \\ &= (\max_x (\mu_{A^*}(x) \wedge \mu_A(x))) \wedge \mu_B(y) \end{aligned}$$

这与模糊控制的结果是一致的。

5.2 神经网络

5.2.1 什么是神经网络

人类利用视觉、听觉、嗅觉、味觉和触觉等感觉器官,从外界获得信息,再通过脑对信息进行处理,然后通过执行器输出到外界。可以说人脑是作为 CPU(中央运算处理装置)的复杂的计算机。在信息处理中,脑神经系统起着重要作用。被称为神经元的脑神经,是由 100 亿个到 1 000 亿个程度不等的细胞形成的神经网络。虽然各神经元的处理速度不快,但是通过大量的神经元进行并行处理,就能够完成人类社会中的复杂判断和行动。

脑神经系统的研究始于 19 世纪。1943 年,麦卡洛克(W. McCulloch)和皮兹(W. Pitts)提出了神经元的数理模型,形成了神经元模型的基础。把神经元模型结合成的多层结构模型,称为神经网络^[6~9]。1958 年,罗森布拉特(F. Rosenblatt)提出了感知器的学习算法,但其学习功能并不充分。1972 年,甘利提出了自组织神经网络,从而在学习理论的解析方面作出了贡献^[9]。1986 年,鲁梅尔哈特(D. Rumelhart)等提出了误差反向传播学习算法。因为误差反向传播算法可以增强学习能力,所以进行了很多应用方面的研究,在模式识别问题、控制问题、预测问题和诊断问题等方面,都有许多应用成果。此外,霍普菲尔德(Hopfield)模型和 LVQ、认识器、玻尔兹曼(Boltzmann)机等神经网络,也都是有应用价值的。本节将以基于误差反向传播学习算法的神经网络为焦点,进行集中地说明。

5.2.2 神经元及其学习功能的研究

神经元由分离的多条输入纤维和一条输出纤维构成,它利用轴突上活动电位的电脉冲传递信息。电脉冲通过所谓突触这种连接体,变换成化学信号,并且传递到下一个神经元(图 5.9)。

麦卡洛克和皮兹提出了把这种神经元的信息处理,作为下列数理模型的研究方法。现在我们设 x_1, x_2, \dots, x_n 为神经元的输入, w_1, w_2, \dots, w_n 为突触的结合加权, x 为神经元的输出。输出 x 变成下一个神经元的输入。神经元的输入与输出的关系可以表示为下列形式(图 5.9):

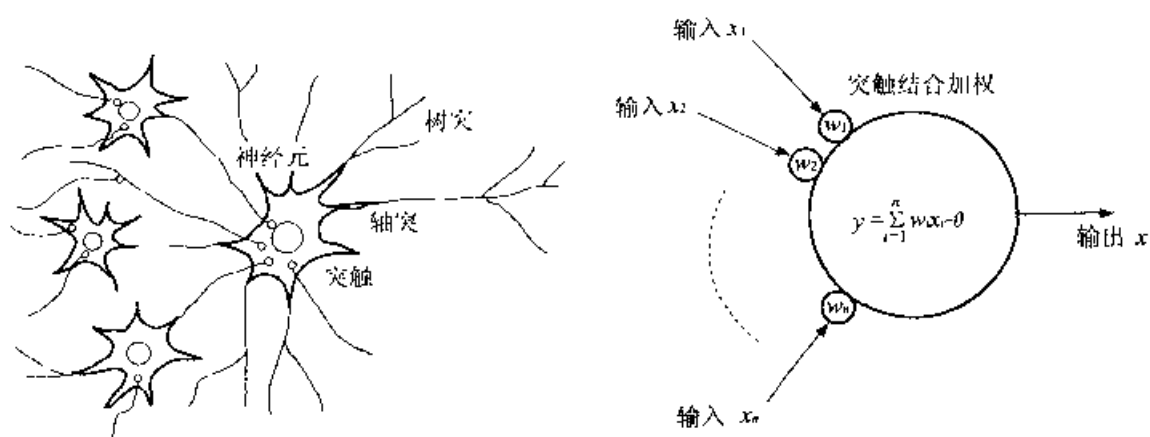


图 5.9 神经元及其数理模型

$$y = \sum_{i=1}^n w_i x_i - \theta \quad (5.31)$$

$$x = f(y) \quad (5.32)$$

式中, $f(\cdot)$ 为输入与输出间的函数。

$$1[u] = \begin{cases} 1 & u \geq 0 \\ 0 & u < 0 \end{cases} \quad (5.33)$$

这是一个阶跃函数。此外, θ 是进行变换时的阈值。

把这种神经元进行多层结合, 就构成了神经网络。

基于神经网络的逻辑运算回路

因为在输入与输出函数为 $1(\cdot)$ 时, 输出为 0 或 1, 所以应用神经网络可以构成逻辑运算回路^[7]。图 5.10 表示了构成逻辑元件的例子。

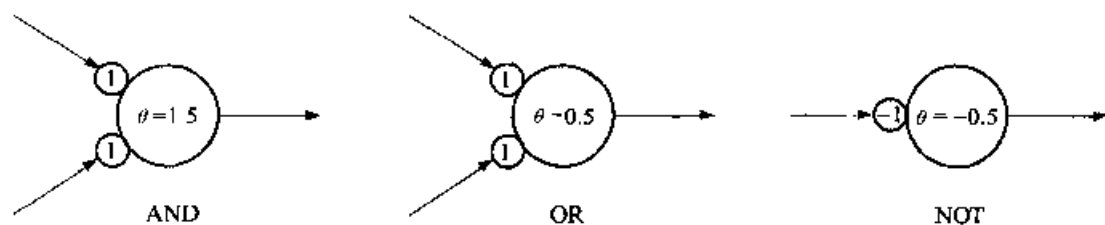


图 5.10 基于神经元的逻辑元件

现在设已知输入 $x_1 = 1, x_2 = 0$ 。在这种情况下, AND 和 OR 的神经元输出, 可以求得如下:

$$\text{AND: } x = 1[1 \times 1 + 1 \times 0 - 1.5] = 1[-0.5] = 0$$

$$\text{OR} : x = 1[1 \times 1 + 1 \times 0 - 0.5] = 1[0.5] = 1$$

另外, 相对于输入 $x_1 = -1$, 可以求得 NOT 的神经元的输出如下:

$$\text{NOT} : x = 1[-1 \times 1 + 0.5] = 1[-0.5] = 0$$

除此以外, 还可以构成触发器等。

下面, 我们来说明关于神经元中的突触结合加权学习。神经元的学法则, 被固定为下列模式:

$$w_i(t+1) = kw_i(t) + \Delta w_i, 0 < k < 1 \quad (5.34)$$

$$\Delta w_i = \alpha \delta x_i, \alpha > 0 \quad (5.35)$$

式中, k 和 α 为表示学习效率的系数, δ 为学习信号, $w_i(t)$ 和 $w_i(t+1)$ 分别表示第 t 次和第 $t+1$ 次时的学习突触结合加权。

在神经元的学习中, 根据如何定义学习信号 δ , 而提出了种种学习方式。合原^[6]对这些学习方式进行了详细说明。赫布(D. O. Hebb)提出了学习只与神经元的输出 x 有关的观点, 称其为赫布的学习。在赫布的学习中, 作出了下列定义:

$$\delta = x \quad (5.36)$$

在赫布的学习中, 当某个神经元处于兴奋的情况时, 为了满足其输出值的需要, 突触结合加权 w_i 将增大。因此, 相对于类似的输入, 其响应会变得比较简单。

另一方面, 罗森布拉特等把从神经元以外得到的外界响应, 看作是神经元的教师信号, 设教师信号 d 与输出 x 的差, 为学习信号 δ :

$$\delta = d - x \quad (5.37)$$

这种学习方法虽然简单, 但是却具有成效。现在让我们考虑教师信号 d 与输出 x 变成 $d \geq x$ 时的情况。在这种情况下, 作为学习, 与输出变大相一致, 应使突触结合加权发生变更。因为 $d - x \geq 0$, 所以对于式(5.35)来说, 当 $x_i \geq 0$ 时, 会变成 $\Delta w_i \geq 0$ (当 $x_i \leq 0$ 时, 则有 $\Delta w_i \leq 0$), 根据式(5.34), 利用 $w_i(t+1)$ 计算出的输出 x 会变得更大。另一方面, 在 $d < x$ 的情况下, 因为 $d - x < 0$, 所以变为 $\Delta w_i < 0$ (或者 $\Delta w_i > 0$), 由于 $w_i(t+1)$ 的输出会变大, 因此, 神经元应进行学习, 使得输出尽量地接近给出的教师信号。作为例子, 考虑下列神经元:

$$x_1 = 0, \quad x_2 = 1, \quad w_1(0) = 0.0, \quad w_2(0) = 0.2,$$

$$\theta = 0.5, \quad k = 0.9, \quad \alpha = 0.2, \quad d = 1$$

这个神经元中的突触结合加权, 进行了下列变更:

($t=0$ 的情况)

$$x = 1[\omega_1(t)x_1 + \omega_2(t)x_2 - \theta] = 1[-0.3] = 0 \quad (5.38)$$

$$\omega_1(1) = k\omega_1(0) + \alpha(d - x)x_1 = 0 + 0 = 0 \quad (5.39)$$

$$\omega_2(1) = k\omega_2(0) + \alpha(d - x)x_2 = 0.18 + 0.2 = 0.38 \quad (5.40)$$

($t=1$ 的情况)

$$x = 1[0 \times 0 + 0.38 \times 1 - 0.5] = 1[-0.12] = 0 \quad (5.41)$$

$$\omega_1(2) = 0.9 \times 0 + 0.2 \times (1 - 0) \times 0 = 0 \quad (5.42)$$

$$\omega_2(2) = 0.9 \times 0.38 + 0.2 \times (1 - 0) \times 1 = 0.542 \quad (5.43)$$

($t=2$ 的情况)

$$x = 1[0 \times 0 + 0.542 \times 1 - 0.5] = 1[0.042] = 1 \quad (5.44)$$

在 $t=2$ 的情况下,神经元的输出变得与教师信号相等。这种学习法则称为**错误订正学习**,错误订正学习采用了感知器的学习。

此外,还提出了下列学习法则:

$$\delta = d \quad (5.45)$$

$$\delta = d - \sum_{i=1}^n w_i x_i \quad (5.46)$$

下面,说明罗森布拉特提出的视觉系统的感知器。图 5.11 示出了感知器的构造。S 层相当于生物体的视网膜,设来自外界的视觉信号为输入量。信号被传递到随后的 A 层。从 S 层到 A 层的突触结合加权,被固定为随机值。A 层被称为连合层,相当于生

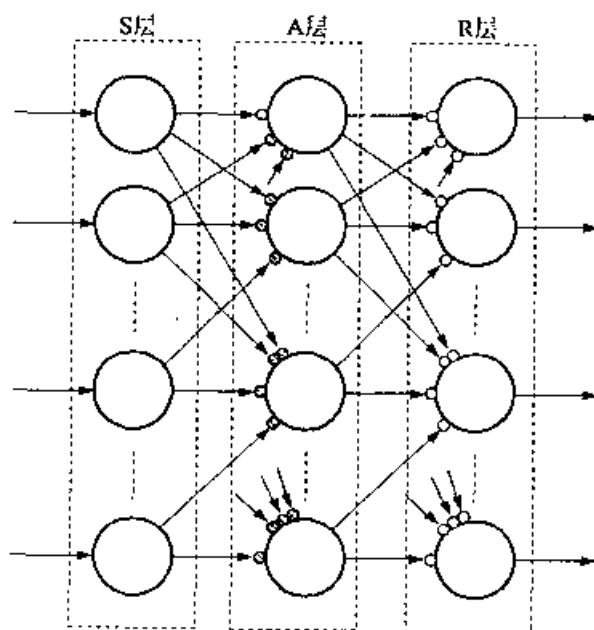


图 5.11 感知器的构造

物体感觉细胞的感受区域。R 层是反应层,它输出信息结果。从 A 层到 R 层的突触结合加权,应用式(5.37)进行学习。例如,设把来自 S 层的字母的图像数据作为输入,那么在 R 层上将构成判别是“A”还是“B”的感知器。如果对学习进行有限次的错误订正学习,那么在它们的模式是线性函数(参考第 6 章)时,就属于可以分离的情况。根据感知器的收敛定理,使得正确地对“A”或者“B”字母的判别成为可能。

5.2.3 误差反向传播学习是一种便利方法

在感知器中,只在输出层的突触结合上进行学习。因此,学习能力低¹⁾。相对于这种方法,在基于鲁梅尔哈特提出的误差反向传播学习的神经网络中,对神经网络中所有的神经元突触结合加权进行学习。误差反向传播学习算法,是一种与最急下降法类似的学习方法。萩原^[8]对适用于误差反向传播学习和最急下降法等算法的区别,进行了说明。

下面对误差反向传播学习进行说明^[8]。在误差反向传播学习中,因为要用到输入与输出函数 f 的微分值,所以函数必须是可微分的。一般来说,下列 S 形函数是有用的(图 5.12):

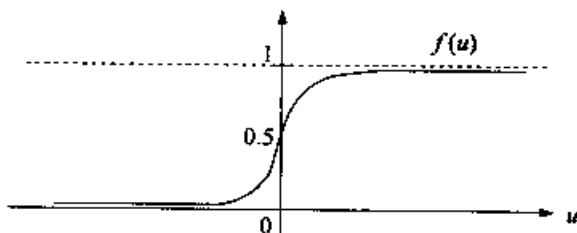


图 5.12 S 形函数

$$f(u) = \frac{1}{1 + \exp(-u)} \quad (5.47)$$

现在我们来考虑图 5.13 表示的神经网络的阶层构造。神经网络的学习依照下列步骤进行:

1) 在 3 层的感知器中,人们已经指出,它不能表示 Exclusive-OR。

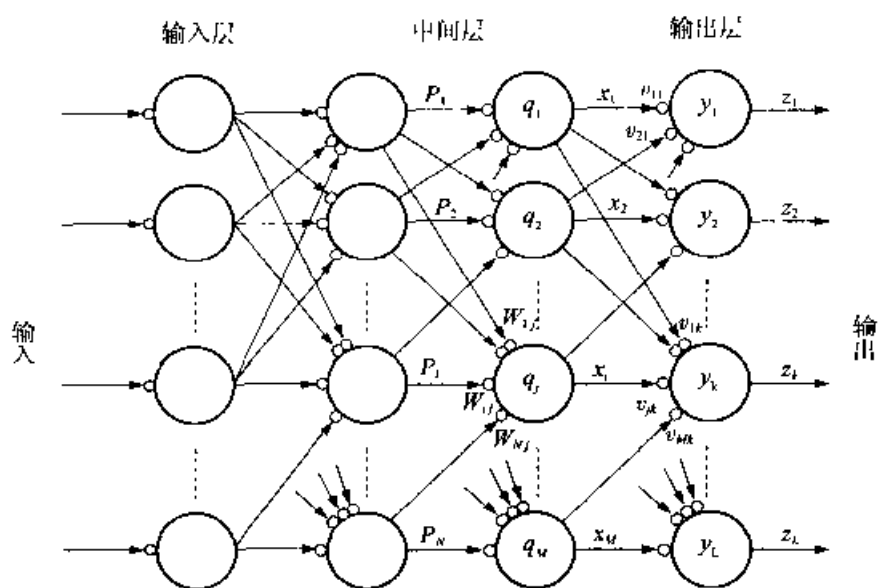


图 5.13 神经网络

【学习算法】

步骤 1 将输入数据输送到输入层。

步骤 2 从输入层到输出层,计算各层中神经元的输出。

• 中间层的场合

$$q_j = \sum_{i=1}^N w_{ij} p_i - \theta, j = 1, 2, \dots, M \quad (5.48)$$

$$x_j = f(q_j), j = 1, 2, \dots, M \quad (5.49)$$

• 输出层的场合

$$y_k = \sum_{j=1}^M v_{jk} x_j - \theta, k = 1, 2, \dots, L \quad (5.50)$$

$$z_k = f(y_k), k = 1, 2, \dots, L \quad (5.51)$$

式中, w_{ij} 和 v_{jk} 分别是中间层和输出层的突触结合加权, q_j 和 z_k 分别是中间层和输出层的神经元输出。

步骤 3 计算输出层中神经元的输出 z_k 与教师信号 d_k 之间的误差平方和。

$$E = \frac{1}{2} \sum_{k=1}^L (d_k - z_k)^2 \quad (5.52)$$

步骤 4 对输出层神经元的突触结合加权进行学习,使得误差平方和 E 达到最小。

$$v_{jk}(t+1) = \eta v_{jk}(t) + \Delta v_{jk} \quad (5.53)$$

$$\Delta v_{jk} = -\alpha \delta_k x_j \quad (5.54)$$

具体作法是,对式(5.54)的突触结合加权变化量 Δv_{jk} 进行下列计算:

$$\begin{aligned}\Delta v_{jk} &= -\alpha \frac{\partial E}{\partial v_{jk}} \\ &= -\alpha \frac{\partial E}{\partial z_k} \frac{\partial z_k}{\partial y_k} \frac{\partial y_k}{\partial v_{jk}} \\ &= \alpha (d_k - z_k) f'(y_k) x_j\end{aligned}\quad (5.55)$$

这里,若采用式(5.47)中的 S 形函数,则式(5.54)中的学习信号 δ_k 变成为下列形式:

$$\begin{aligned}\delta_k &= (d_k - z_k) f'(y_k) \\ &= (d_k - z_k) f(y_k) (1 - f(y_k))\end{aligned}\quad (5.56)$$

步骤 5 对中间层神经元的突触结合加权进行学习,使得误差平方和 E 达到最小:

$$w_{ij}(t+1) = kw_{ij}(t) + \Delta w_{ij} \quad (5.57)$$

$$\Delta w_{ij} = \alpha \delta_j p_i \quad (5.58)$$

与式(5.55)的情况相同,对式(5.58)的突触结合加权变化量 Δw_{ij} 进行下列计算:

$$\begin{aligned}\Delta w_{ij} &= -\alpha \frac{\partial E}{\partial w_{ij}} \\ &= -\alpha \frac{\partial E}{\partial z_k} \frac{\partial z_k}{\partial x_j} \frac{\partial x_j}{\partial q_i} \frac{\partial q_i}{\partial w_{ij}} \\ &= -\alpha \sum_{k=1}^L \frac{\partial}{\partial z_k} \left(\frac{1}{2} \sum_{k=1}^L (d_k - z_k)^2 \right) f'(y_k) v_{jk} f'(q_i) p_i \\ &= \alpha \sum_{k=1}^L (d_k - z_k) f'(y_k) v_{jk} f'(q_i) p_i \\ &= \alpha \sum_{k=1}^L \delta_k v_{jk} f'(q_i) p_i\end{aligned}\quad (5.59)$$

这里,若采用式(5.47)中的 S 形函数,则式(5.58)中的学习信号 δ_j 变成为下列形式:

$$\begin{aligned}\delta_j &= \sum_{k=1}^L \delta_k v_{jk} f'(q_j) \\ &= \sum_{k=1}^L \delta_k v_{jk} f(q_j) (1 - f(q_j))\end{aligned}\quad (5.60)$$

步骤 6 在中间层上进行的神经元突触结合加权学习,直到输入层,要反复地进行。

步骤 7 对于新的输入数据,重复进行上述的步骤(1)~(6)。

根据式(5.55)和式(5.59)可以清楚地看出,突触结合加权的

修正量,现在变成了 $\alpha \times \text{误差} \times f'(\quad) \times \text{输入值}$ 。因此,误差和神经元的输入值越大,则突触结合加权的修正量便变得越大。

根据误差传播学习,当给定的神经网络输出等于教师信号时,所有神经元的突触结合加权就都得到了修正。对于从输入层到输出层进行的信号传递处理,突触结合加权的的学习是从输出层到输入层反向传播的。因此,称之为误差反向传播学习(反向传播学习)。

误差反向传播学习算法,在语音识别、手写文字识别、图像识别等领域,都取得了很多应用成果,在控制、预测、诊断等方面的应用,也正在试探性地进行着。现在我们来介绍一个利用误差反向传播学习,构成模糊规则的事例。让我们由基于输入变量 x_1, x_2, \dots, x_n 和输出变量 y 的输入-输出数据 $(x_i^*, y_i^*) = (x_{i1}^*, x_{i2}^*, \dots, x_{in}^*, y_i^*)$, $i=1, 2, \dots, N$, 构成模糊规则。首先,确定事件前部的模糊集合。根据群聚方法等,可以将输入数据 $x_i^* = (x_{i1}, x_{i2}, \dots, x_{in})$ 分解成许多群 C_s , $s=1, 2, \dots, r$, 隶属于各群 C_s 的程度 Z_s 定义如下:

$$Z_s = \begin{cases} 1 & : x_i^* \in C_s \\ 0 & : x_i^* \notin C_s \end{cases} \quad (5.61)$$

隶属程度 Z_s 表示了数据 x_i^* 隶属于各 C_s 的可能性情况。其次,将 $x_i^* = (x_{i1}, x_{i2}, \dots, x_{in})$ 分配到神经网络的输入层,将可能性的程度 $Z_i = (Z_{i1}, Z_{i2}, \dots, Z_{ir})$ 分配到输出层。误差反向传播学习结束后的神经网络,表示了隶属于输入 x_i 与 C_s 的可能性程度之间的关系,对于学习后的神经网络的输入数据 x_k^* , $k=1, 2, \dots, K$, 设其输出为 $Z_k^* = (Z_{k1}^*, Z_{k2}^*, \dots, Z_{kr}^*)$ 。另外,由模糊规则的事件前部的模糊集合 F_{v_j} , $j=1, 2, \dots, n$, 构成了输入空间的 n 维模糊集合,设该集合为 F_s 。模糊集合 F_s 的隶属值 $\mu_{F_s}(x_k^*)$ 定义如下:

$$\mu_{F_s}(x_k^*) = Z_{ks}^*, k=1, 2, \dots, K \quad (5.62)$$

图 5.14 示出了决定事件前部模糊集合的过程。另外,图 5.15 表示了学习后隶属值的例子。

另一方面,利用包括事件后部各规则的神经网络进行辨识。在图 5.16 中,表示了模糊规则的神经网络,作为事件前部神经网络输出值的隶属值 $\mu_{F_s}(x_i^*)$, $s=1, 2, \dots, r$, 表示了式(5.29)中的 g_s 的值。另外,事件后部神经网络的输出值是式(5.28)线性方程式的估计值 y_s^* , b^* 表示式(5.30)的推理结果。

这种模糊规则与神经网络的融合模型,称为模糊神经网络。

络^[10~12]。现在已经提出了许多种模糊神经网络的模型,应用在以控制和模式识别为中心的领域内。

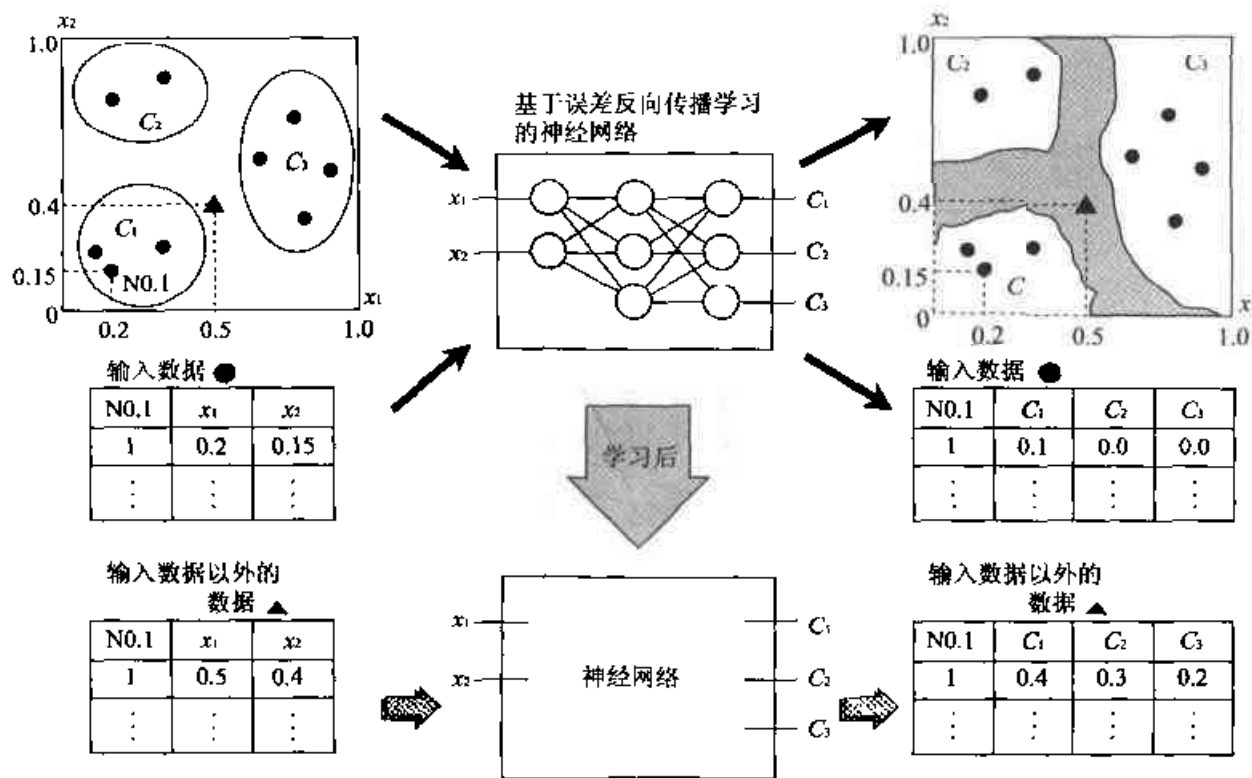


图 5.14 事件前部模糊集合的确定过程

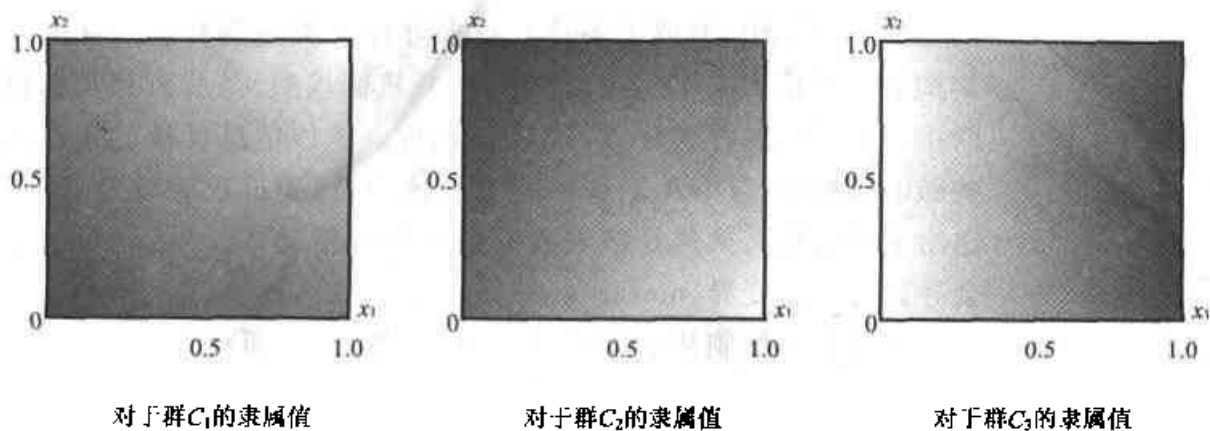


图 5.15 事件前部的隶属值的例子

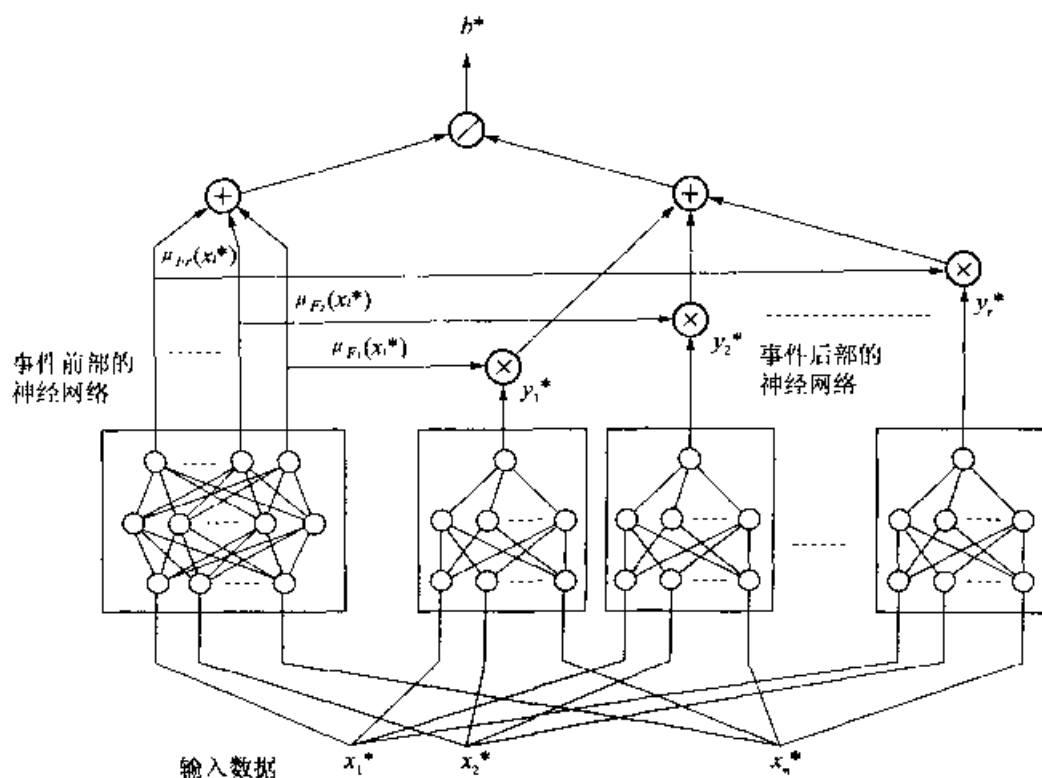


图 5.16 表示模糊规则的神经网络

5.3 遗传算法

5.3.1 什么是遗传算法

遗传算法(genetic algorithm, GA),是在生物进化的启示下,得到的一种搜索和自适应算法。生物在其产生以来长达 40 亿年的“进化”过程中,形成了多达一百多万种各类生物种群。在双螺旋构造的碱排列中,著名的 DNA(脱氧核糖核酸)是所有遗传信息的携带者。碱的种类有 A,T,C,G 四种。遗传信息就被记载在这些碱的排列中。DNA 记载了所有遗传信息,通过正确地复制,传送给子孙后代。虽然复制失败只是在少数情况下发生,但是这却会导致突然的变异(mutation)。

增殖是单细胞生物那样的低等生物,通过细胞分裂进行的。在高等生物中,增殖是通过有性生殖进行的。这时,将会继承来自双亲的遗传信息,但是对于同一个孩子,把双亲的遗传信息完全混杂在一起是做不到的,总有一点点不同保留于子孙后代身上。实际上,在被称为减数分离过程中进行的是染色体的交叉(cross-

over), 有性生物正是利用这种交叉, 获得了更大的进化能力。

对于 GA 来说, 用以前的数理方法进行求解是一个困难的问题, 但是可以这样求解这个问题, 这就是可以把这个问题的候补解视为生物个体, 而生物个体对环境有很高的适应能力, 这样, 应用进化机构, 就能够获得提供出更好解答的候补解。将生物进化能力应用于信息处理的试探性研究工作, 始于 20 世纪 60 年代。

具有代表性的各种进化论学说

达尔文的进化论不能说明所有的进化问题。应用最近发展起来的分子生物学, 使得各种各样的新事实变得明朗化了。以下是一些具有代表性的学说。

- **拉马克进化学说** 如果经常使用身体上的特定部分, 则该部分就会发达, 反之如果不经常使用, 该部分就会退化, 且这样获得的遗传特征可以遗传。这称为用不用学说。现在对这种学说一般持否定观点。

- **新达尔文主义** 主张自然选择(淘汰)的达尔文名著《物种起源》, 在其再版中, 关于对获得遗传特征的遗传的认识也发生了变化。它是一种只承认纯粹的自然选择的进化学说。

- **综合进化学说** 以达尔文的自然选择作为基础, 在突然变异、杂交、隔离等方面取得的种群遗传学成果基础上, 进行综合性说明。在现代的 GA 中, 这种考虑方法被认为是最基本的方法。

- **中立进化学说** 遗传基因的进化, 不仅仅是因为自然选择而引起的, 对于生物来说, 不论是否有利, 中立的突然变异积累, 会起到重要作用。这就是所谓中立进化学说。

- **连续共生学说** 一些同类细胞进行合并, 共同产生出更新的细胞。真核细胞内的叶绿体和线粒体等已为人们所熟知。

- **病毒进化学说** 遗传基因通过病毒从一种生物运送到另一种生物。这种被运送的遗传基因, 使生物的遗传基因发生了变化。

1975 年密歇根大学的霍兰(Holland)出版了他的著作《Adaptation in Natural and Artificial Systems》, 从而确立了 GA 的基础。德琼(DeJong)对作为最优化方法的 GA 进行了评估, 从而确认了这种方法的有效性。同样, 戈德堡(Goldberg)从理论到煤气管道的配置规划等应用领域, 进行了广泛地研究, 并且于 1989 年出版了他的名著^[21], 该书现在已成为 GA 研究的经典著作。1985 年, 召开了第一届 GA 国际会议(International Conference on Ge-

netic Algorithms, ICGA)。现在已经创办了 GECCO (Genetic and Evolutionary Computation Conference), CEC (Congress on Evolutionary Computation), PPSN (Parallel Problem Solving from Nature) 等国际会议, 活跃的国际会议在不断地进行着。

5.3.2 单纯 GA 的基本步骤

在 GA 中, 考虑个体(individual)的集合, 即考虑种群(population)的进化问题。个体的特性用染色体(chromosome)来确定。在实际的生物中, 由 DNA 构成实际的物理状态。在 GA 中一般都采用比特系列, 但是也可以采用数值列和任意的符号列。在染色体的各个位置中, 记载着遗传信息, 这样的位置称为遗传基因座位(locus)。对于各遗传基因座位, 决定其遗传特征的代码称为遗传基因(gene)。各个个体的染色体上的遗传基因组合内部表示, 称为遗传基因型(genotype), 而根据遗传基因型形成的各个个体的遗传特征, 则称为表型(phenotype)。在这种情况下, 根据对遗传基因型的种种解释, 可以自由地设定表型。根据这种表型可以确定适合度(fitness)。在种群内, 适用于各个个体遗传操作(genetic operator)的新的个体产生了出来, 于是按照其适合度进行了选择(selection, 又称为淘汰), 并实现了更新换代(generation)。以下针对所谓单纯 GA (simple GA; SGA) 的最基本的 GA 步骤进行说明(参考图 5.17)。

步骤 1 初生代种群的生成(initialization)。

作为初生代的种群, 产生了 N 个具有随机染色体的个体。在比特系列的情况中, 随机地生成了由随机数决定的长位串。设世代 $t=0$ 。

步骤 2 评估(evaluation)。

根据各个个体的遗传基因型求表型, 依照预先确定的方法对各个个体 X_i 进行评估, 并且求出各自的适合度 $f(X_i)$, 检查是否满足终止条件。如满足终止条件则结束, 如果不满足终止条件则依照下列步骤继续进行。所谓终止条件, 就是在种群中, 满足预先确定的标准的个体已经出现, 世代交替的数值达到预先确定值的情况等已经发生。

步骤 3 选择(selection)。

从当代 t 的 N 个个体 $X_1 \sim X_N$ 中选择出下一代 $t+1$ 的 N 个个体, 选择时允许重复选择。在选择各个个体 X_i 时, 假设选择概

率为

$$p(X_i) = \frac{f(X_i)}{\sum_{j=1}^N f(X_j)} \quad (5.63)$$

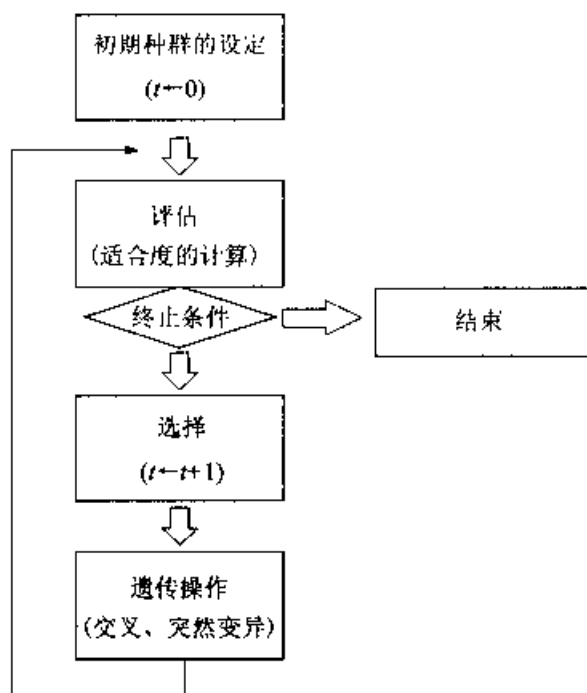


图 5.17 单纯 GA 的步骤

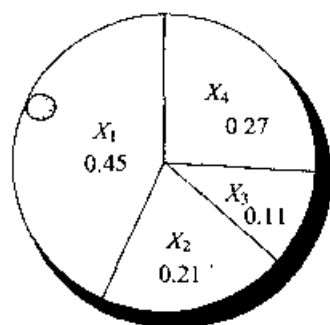
在上式中,右边的分子是个体 X_i 的适合度,分母是当代种群的适合度之和。也就是说,各个个体在下一代的生存可能性与其在种群中的自身的适合度成比例。这种选择方式被称为比例选择方式(proportional selection)。在比例选择方式中,与适合度成比例,从而保留了对自己的仿制,适合度越高的个体,被选作下一代个体的概率越大,适合度小的个体被淘汰的概率会变大。在理解比例选择时,可以设想一个图 5.18 所示的轮盘,它是根据各个个体的适合度做出来的,是一种仅以选择的个体数(N)环绕着轮盘作出的图像,很容易理解。个体 i 被选作下一代个数的期望值 N_i 可根据式(2.63)求出为

$$N_i = \frac{f(X_i)}{\sum_{j=1}^N f(X_j)} \times N = \frac{f(X_i)}{\sum_{j=1}^N f(X_j)/N} = \frac{f(X_i)}{\bar{f}} \quad (5.64)$$

式中, \bar{f} 为种群的平均适合度。

个体 X_i	适合度 $f(X_i)$	选择概率 $p(X_i)$	选择个体数 的期望值 $p(X_i)/\bar{f}$	被选择的 个体数(例)
X_1	150	0.45	1.62	2
X_2	80	0.21	0.86	0
X_3	40	0.11	0.44	1
X_4	100	0.27	1.08	1

$$\sum_{i=1}^4 f(X_i) = 370, \bar{f} = 92.5$$



轮盘

图 5.18 比例选择方式

步骤4 遗传操作的应用。

设 $t \leftarrow t + 1$, 对于这个新的一代, 应用遗传操作。在遗传操作过程中, 有交叉(crossover)和突然变异(mutation)发生。所谓交叉, 就是由 N 个个体随机地生成 M 组对, 如图 5.19 所示, 通过随机地设定两个个体的染色体间位置的交叉, 在交叉位置处重新组成了遗传基因的类型。由这种操作而得到的两个子个体更换了原来的双亲个体。因此, 在继承了双亲的遗传特征后, 生成了新的个体。发生交叉的概率称为交叉率(crossover rate)。突然变异如图 5.20 所示, 这时对随机选择的遗传基因座位上的值进行了强制性变更。对各位置施加突然变异的概率, 称为突然变异率(mutation rate)。

双亲个体1	1010011011
双亲个体2	0101110010
	↓
子个体1	1010010010
子个体2	0101111011

图 5.19 交叉

突然变异前	1010010010
	↓ ↓
突然变异后	1110010010

图 5.20 突然变异

以上由步骤2到步骤4构成GA的一个周期, 称之为代(generation)。通过反复进行世代交替, 在种群内的个体中, 适合度高的新的个体就会缓慢地增加起来。

5.3.3 简单函数最优化举例

在这里, 为了理解SGA的运行, 在区间 $[-6.4, 6.3]$ 上求单变量简单函数

$$f(x) = e^{-10 \cdot 2|x|} \cos^2 \pi x / 2 \quad (5.65)$$

的最大值中,进行应用 SGA 的实验。当然,在全解 $x=0$ 时,其函数值为 1。然而由图 5.21 可以清楚地看出,因为这个函数有许多局部峰值,所以存在着不利的一面,这就是在应用梯度法等其他一些方法时容易陷入局部解中,因此很难找到最优解。为了进行对变量 x 的实数值的表示,用 7 比特的比特列表示个体,并设其为遗传基因型。这种比特列被看作是纯二进制数。若设这种纯二进数的

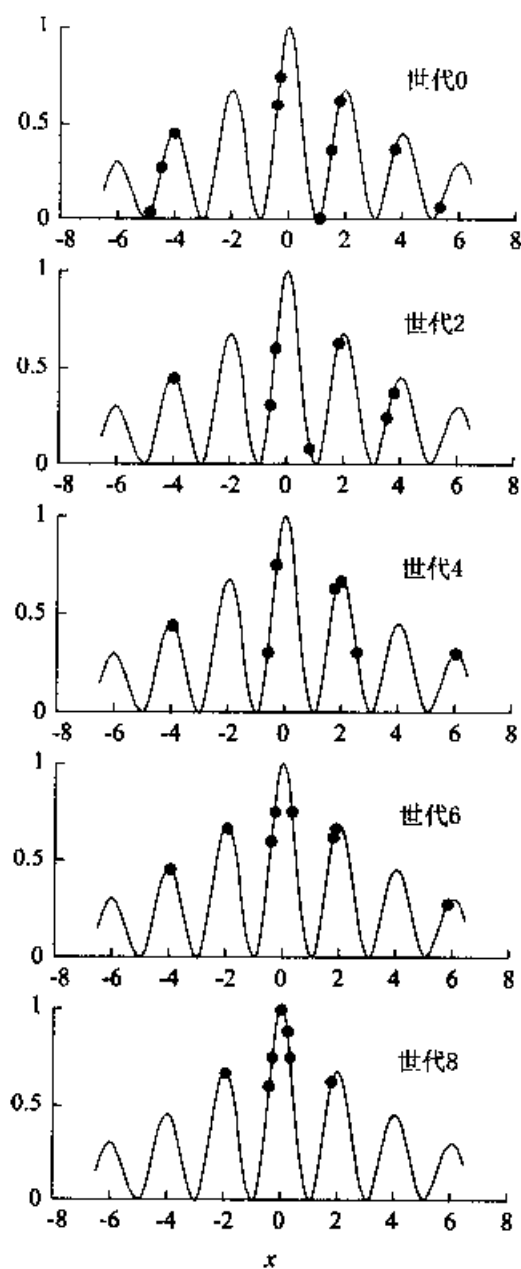


图 5.21 种群的个体分布状况的变化

值为 x' , $x'=0$ 与 -6.4 相对应, $x'=2^7-1$ 与 6.3 相对应。于是作为从遗传基因型向表型参数 x 的变换,可以得到下列关系式:

$$\begin{aligned} x &= -6.4 + x' \times (6.3 - (-6.4)) / (2^7 - 1) \\ &= -6.4 + 0.1 \times x' \end{aligned} \quad (5.66)$$

在这种情况下,位串 $\langle 0000000 \rangle$ 和 $\langle 1111111 \rangle$ 分别表示 -6.4 和 6.3 , 参数 x 取 0.1 刻度的值。这里,设各个个体的适合度为其函数值 $f(x)$ 。因此,SGA 可以期望实现这样一种运作,在这种运作下,可以搜索到使 $f(x)$ 达到最大的一点,即搜索到 $x=0$ 这一点。因为 $x=0$ 对应于 $x'=64$, 所以给出最优解的个体的位串是 $\langle 1000000 \rangle$ 。这里设个体数 $(N)=10$, 交叉率 $=0.6$, 突然变异率 $=0.006$ 。在这个实验中,经过 8 个世代,得到了最优解 $x=0$ 。各世代种群的个体分布状况表示在图 5.21 上。从随机地生成初期种群开始,随着世代的更叠,一方面使分布于最优解领域内的个体增加,另一方面种群会进入收敛状态。表 5.1 表示了各世代种群的最好个体适合度与平均适合度的变化情况。

表 5.1 种群适合度的变化

世 代	最好个体的适合度	平均适合度
0	0.74766	0.35805
1	0.63105	0.46267
2	0.63105	0.43840
3	0.74766	0.55454
4	0.74766	0.54519
5	0.74766	0.61764
6	0.74766	0.61764
7	0.86904	0.68785
8	1.00000	0.76311

5.3.4 单纯 GA 的扩张

从对进化过程进行模拟的意义上看,GA 是一种缓慢进行的结构,它可以采用种种不同的构成方案。对于实际的生物进化来说,由于分子生物学的进步,使人们获得了许多新的知识,因此,也可以说进化论现在达到了百花齐放,争奇斗艳的局面。因为 GA 在工程上利用了“进化”的力量,所以实际的生物进化怎样进行是

不必进行限制的。

下面,在加快 GA 收敛时,为了有效地实施谋求种群多样化这种进化过程,我们将简单地涉及一些重要的基本扩张方式,关于这方面的详细内容,希望读者参考书后面列出的有关文献。

1. 选 择

(1) **定标** 设适合度已被确定,没有必要让这个值反映原来选择时的概率,引入某些函数,可以使适合度的差别扩大或者缩小。引入这样一些函数的工作,称之为定标。作为定标的方法,虽然已经提出了各种各样的方法,但是最简单的方法还是设定个体 X_i 的适合度为

$$f'(X_i) = f(X) - f_{\min} \quad (5.67)$$

时的方法。 f_{\min} 是当前这一代最差个体适合度的值。通过用 $f'(X)$ 代替 $f(X)$ 可以扩大个体间适合度的相对差值。

(2) **等级法** 在等级法中,从适合度高的个体开始,对每一个个体附加上等级,并且对于各个等级都会以事先确定的概率遗传给子孙后代。在等级法中,即使在各个体间适合度接近,各个体间的选择概率差仍可以得到维持。

(3) **良种保存法** 即使在某一代中出现了好的个体,由于交叉和变异,也会使其遭到破坏。在良种保存法中,如果前一代的最优个体和具有同样构造的东西,在现在这一代中不存在,那么就把它作为当代的第 $N+1$ 号个体加进这一代中。这种方法,虽然在使优良个体的遗传基因,在种群中迅速扩展有比较高的可能性时,存在着陷入局域解的危险,但是在提高进化效率中,它仍不失为一种有效的方法。

2. 遗传的操作

(1) **多点交叉** (multi-point crossover) 在 SGA 中,介绍了一点交叉 (one-point crossover),在多点交叉中设置了多个交叉位置。图 5.22 表示了一个两点交叉的例子。一般情况下,多采用两点交叉的方法。

(2) **均匀交叉** (uniform crossover) 均匀交叉就是设定交叉时的随机模式作为掩码。在这种掩码模式“1”的位置上,对其遗传基因座位上的遗传基因进行交换。图 5.23 表示了均匀交叉的例子。

(3) **其他交叉** 过去,我们以位串的情况作为前提,针对染色体的遗传操作进行了阐述,在 GA 中,每个问题领域中都有各种各

双亲个体1	10 10011 011
双亲个体2	01 01110 010
↓	
子个体1	1001110010
子个体2	0110011010

图 5.22 两点交叉

双亲个体1	1010011011
双亲个体2	0101110010
↓	
子个体1	1100011010
子个体2	0011110011

图 5.23 均匀交叉

样的染色体表示,即存在着编码法。例如,在确定进程表问题的顺序时,一般对进程中的各项工作都赋予一连串的符号,并且把这些符号排列起来,以表示各项工作的进行顺序。对于这种形式的染色体表示,在利用单纯地交换位置,实施遗传基因交互插入这种交叉方式时,在子个体的染色体中会出现重复的符号,从而形成了对象问题无解的后代(参见图 5.24)。作为能解决这个问题的交叉方法,现在已经提出了部分一致交叉(partially matched crossover, PMX)、顺序交叉(order crossover, OX)、周期交叉(cycle crossover)等各种交叉方法。

图 5.25 表示了顺序交叉(OX)的例子。在顺序交叉中,首先是两个交叉位置间的遗传基因互相交换插入。然后从双亲个体1的第2个交叉位置开始,消除掉被交换插入的遗传基因(图 5.25 中的 E,G,I,B),作出排列 H-A-C-D。从第2个交叉位置开始,对这个排列依次进行配置,即可生成子个体1。子个体2也可以用同样的方法生成。

双亲个体1	A B C D E F G H I
双亲个体2	C D E G I B A F H
↓	
子个体1	A B E G I B G H I
子个体2	C D C D E F A F H

图 5.24 顺序表示交叉的失败

双亲个体1	A B C D E F G H I
双亲个体2	C D E G I B A F H
↓	
子个体1	D F E G I B H A C
子个体2	I B C D E F A H G

图 5.25 顺序表示的 OX

在 5.3.3 节的例子中,曾经把位串用到染色体的表示中,解决了函数的最优化问题。实际上,把实数值列作为其原来的染色体代码,解决上述最优化问题也是可行的。这样构成的 GA,也称为实数码 GA(real-coded GA)。针对实数码 GA 也提出了各种交叉方法。图 5.26 就是一个以实数码的交叉为代表的混合交叉(BLX- α)的例子。在 BLX- α 中,从被定义在双亲周围的附近区域内(图 5.26 的粗线框内),随机地选择两点,并使其生成为子个

体。作为 α 的值,可以采用 0.5。另外,在许多 GA 中,一般都应用了两个亲个体(双亲)这种与有性生物相同的交叉方法,对于三个以上亲个体进行交叉的效果研究也在进行中。

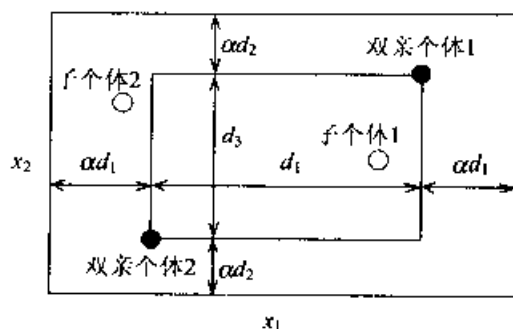


图 5.26 实数码交叉 BLX- α

3. 编 码

采用什么样的染色体表示,是 GA 应用研究的重要课题。在用位串进行数值表示时,一般都采用格雷码(gray code)¹⁾。

4. 分散 GA

一般来说,在种群的规模比较小的情况下,该种群的进化会趋于收敛,而且收敛后其适合度略有提高。因此,在分散 GA 中,设置许多小种群,并且使这些小种群并行运行。在小种群之间,使其进行个体的定期“移民”(immigration)。这样做的结果,就可以在整体上创建出实现性能非常高的 GA 的局面。

5.3.5 模式定理

虽然对 GA 工作原理的分析进行了许多尝试,但是目前还是不充分的。由霍兰提出的模式定理(schema theorem),用数学方法说明了 GA 的原理,从而揭示了许多背景材料。

当用一维的文字列表示染色体时,这其中意味着会有某种形态发生,这种形态被称为模式。模式定理说明,由于世代的交替,模式将会产生怎样的变化。当个体用长度为 L 比特的文字列表示时,模式是用由“0”,“1”,“*”构成的长度为 L 的文字列表示的。“*”是 don't care 的符号,表示可以是“0”,也可以是“1”。例如

$$H = *10*0$$

1) 相邻数值的比特模式具有只差 1 比特的特性。

这种模式,它表示了下列四种文字列:

01000,01010,11000,11010

这时,称这些文字列被“包含”在 H 内。这样,模式就表示了搜索空间的部分空间。首先,对模式我们来定义它的定义长(defined length) $\delta(H)$ 和位数(order) $o(H)$ 。模式的定义长,是指从左方开始见到的“*”以外的文字,到最后的“*”以外的文字之间的距离。例如, $\delta(*10*0)=3$, $\delta(*10**)=1$ 。模式的位数,是指“1”和“0”,即非“*”文字的数目。例如, $o(*10*0)=3$, $o(*10**)=2$ 。

设在第 t 代种群中存在着模式 H , 包含在模式 H 内的个体数为 $m(H, t)$, 包含在模式内的个体的适合度平均值为 $f(H, t)$ 。这时若假设不适合交叉和突然变异, 只能进行比例选择, 则在第 $t+1$ 代内存在的模式 H 中, 包含的个体数期望值可以用下式表示:

$$m(H, t+1) = m(H, t) \cdot \frac{f(H, t)}{f} \quad (5.68)$$

当考虑交叉时, 定义长的长模式遭到破坏的可能性增大, 若设交叉率为 p_c , 则其概率为 $p_c \delta(H) / (L-1)$ 。同样, 位数大的模式, 由于突然变异, 破坏的可能性也会增大, 若设突然变异率为 p_m , 则其概率为 $o(H) p_m$ 。因此, 当考虑交叉和突然变异时, 式(5.68)变成下列不等式:

$$m(H, t+1) \geq m(H, t) \cdot \frac{f(H, t)}{f} \cdot \left[1 - p_c \frac{\delta(H)}{L-1} - o(H) p_m \right] \quad (5.69)$$

在这里变为不等式, 不仅破坏了模式 H , 而且由于交叉和突然变异还生成了新的内容。式(5.69)被称为模式定理, 它阐明了 GA 的基本原理。这就是说, “定义长是短的, 位数是小的, 而且适合度处于平均值以上的模式, 其下一代的存活率会增大”。这种模式被称之为积木(building block)。在 GA 中, 人们正在研究, 用任何这样一种积木去灵活地进行组合, 并从总体考虑去生成具有高适合度的个体, 这种考虑方法被称为积木假说。

5.3.6 遗传算法的应用

在进行最优化设计或规划时, 有许多问题不能用数理方法求解, 各种近似解法正在研究之中。GA 与过去人们知道的近似解法比较, 一般还不能说更为优越, 但是以“进化能力”作为解题基点的 GA, 与过去的方法相比, 有着很大的区别。这种方法不会深入地涉及问题的构造, 所以当给定适合度时, 随后就可以利用进化

能力,比较简单地求解问题,从而具有良好的性能/价格比。再加上它是一种摆脱了以往的困难的问题求解方法,所以可以期待它将获得广泛应用。在应用 GA 时,必须经过下列一些步骤:

(1) 关于对象问题,设定能实施最小化或最大化的评价函数(目标函数)。

(2) 对于问题的候补解,决定用遗传基因型表示(编码)方法,反之,从遗传基因型返回到候补解,决定用解码方法。

(3) 制定由目标函数确定适合度的相应规则。

(4) 适当地选取交叉,突然变异等的遗传算子和选择法,或者重新进行设计。

(5) 适当地设定种群的规模和交叉率等各种参数。

虽然 GA 具有不深入涉及对象问题构造的特征,但是因为上述步骤依赖于对象问题,所以必定会产生试行错误。

在表 5.2 中,列举了 GA 的代表性应用领域和例子。

表 5.2 GA 应用举例

设计问题及计划	印刷线路板的最佳位置 LSI 的布局设计 神经网络的合成 通信网的动态预测路由选择 喷气发动机的设计 空气压缩控制机的设计 硬件进化 购买计划问题 通信网络的设计 透镜系统的设计
控制问题	电梯群管理 煤气管道控制 过程控制 配电系统的损失最小化控制
程序化问题组合最优化问题	加工车间的程序化 各种工程的程序化 产品投产的程序化 公共汽车运行时刻表的编制 流动推销员问题
其他问题	DNA 排列的信号图提取 DNA 的构象分析 分子系统树的绘制 图像分析和复原 概念形成问题 战略获得问题 故障诊断 蛋白质的构造估计

5.3.7 遗传算法的一些同类方法

前面我们在生物进化的启发下,对作为搜索方法的 GA 进行了阐述,下面我们简单介绍把生物作为模型的其他一些研究。人们把包含 GA 的进化作为关键词的一些研究称之为“进化计算(evolutionary computation,EC)”。近年来,这些研究之间的相互交流日益频繁,现在以 ICGA 为先驱的许多国际会议和杂志,都把“进化计算”作为共同的关键词使用了。

1. 进化策略(evolutionary strategy,ES)

所谓 GA,在欧洲独立地把进化作为模型的 ES 研究,始于 20 世纪 60 年代,是由雷切伯格(Rechenberg)和施韦菲尔(Schwefel)进行的。对于 GA 对各种各样染色体表示的处理,ES 把求解以实数空间作为定义域的非线性最优化问题作为其主要目的,它与 GA 以交叉作为其主要遗传操作,以突然变异作为遗传操作的中心等,是不同的。

2. 进化编程(evolutionary programming,EP)

EP 为使有限状态自动机进化的构架。EP 是 20 世纪 60 年代,由福格尔(L. Fogel)等人首先提出来的,其历史与 GA、ES 同样悠久。根据福格尔(D. Fogel)的研究工作,进行了在实数函数最优化问题方面的应用工作,作为与 ES 类似的方法得到了发展。

3. 遗传编程(genetic programming,GP)

在人工智能中,在知识表示中往往要采用图构造和树构造。GP 扩展了 GA 的遗传基因型,它是一种能处理图构造(特别是树构造)的方法,这种方法是由科扎(J. Koza)提出的。20 世纪 90 年代,对 GP 的研究开始活跃起来,现在正在对程序的自动生成和机器人的行为学习等,进行着广泛的应用研究。

练习题

1. 试描绘自己的“青年”、“中年”和“老年”的隶属函数。
2. 试利用隶属函数描述轮廓分明的集合 $A=[30 \text{ 岁以上}, 40 \text{ 岁以下}]$
3. 试对练习题 1 中构成的模糊集合“青年”的补集进行描述。另外,对“中年”和“老年”的交集进行描述。
4. 表示“小的”模糊集合 A,与表示“大的”模糊集合 B,由下列方式构成:

$$A=0.9/1+0.7/2+0.4/3+0.1/4$$

$$B=0/1+0.2/2+0.8/3+1/4$$

这时,试求 $\overline{A \cap B}$, $A \cup \overline{A}$, $A \cap \overline{A}$ 。

5. 试针对浴池内的“热水”、“正合适的水”、“温水”、“大约 40℃ 的水”等情况,描绘它们的模糊数。
6. 试用中文叙述模糊规则的第二项。
7. 设您在驾驶汽车的过程中,为了能使汽车停在停车线上,试构成两条模糊规则。假设输入变量取车的速度 v 和到停车线的距离 d ,输出变量为刹车力 b (设未对闸施加作用时的刹车力为 0,对闸施加作用时的最大刹车力为 10)。
8. 在练习题 7 的模糊规则中,当被输入的量 $v^* = 60\text{km/h}$, $d^* = 20\text{m}$ 时,试利用模糊控制求最终的刹车力。
9. 设 $x_1 = 0$, $x_2 = 1$, $w_1(0) = 0.0$, $w_2(0) = 0.2$, $\theta = 0.5$, $k = 0.8$, $\alpha = 0.2$ 。试确定在神经元的输出 $t = 3$ 时,等效于教师信号 $d = 1$ 。
10. 在表 5.1 中,平均适合度在第二代和第四代都减少了。试分析导致这种情况的原因。
11. 用图 5.24 作出的子个体不能成为候补解,试阐明其理由。对于 PMX 和 CX,试以文献为基础,对交叉结构进行分析。
12. 虽然模式定理是 GA 的重要基础定理,但是它不能完全说明 GA 的运行。试分析对这个定理的哪一方面,有必要作进一步地研究。

第 6 章

模式识别

在现实世界中,人类发出的声音、印刷或者书写出的文字、眼睛看到的风景,以及测量器输出的信号等,以各种各样的模式存在着。如果计算机能够听出并且分辨出这些模式,就能够简单地输入到计算机,并且可以减轻专业人员的作业负担。把模式输入到计算机后,为了对其实施辨识,需要进行怎样地处理呢?在本章中,将学习模式识别的基本考虑方法,并且以声音的识别作为例子,学习模式识别技术的应用。

6.1 什么是模式识别

正如本书前面所述,用计算机来实现人类的智能活动受到关注,各种各样的有关人工智能的研究正在进行。人类的智能活动,大部分是以语言为主的记忆,推理等离散符号的处理这种形式进行的。另外,由于现在的计算机擅长于对符号进行处理,所以可以说多数的人工智能研究都是以符号处理为基础进行的。但是,人们接触到的外界信息,除了符号以外,还存在着以图像和声音这种连续量表示的模式,所以人们自然地会想到,对于各种模式,采用赋予一定意义的符号化方法。

对应于各个符号的模式的集合称为类,对于被输入的模式,确定其所属类别的问题,就是模式识别。基于这种考虑,人们把图像和声音之类的模式,变换成赋予一定意义的符号。模式识别一方面将包含着模式信息的含义提取出来,同时它作为高级信息处理系统的预处理部分,根据提取出的含义,完成适当的复原响应,不仅如此,它还具有自动地识别书写在明信片上的邮政编码等的的能力,这种自身的自动识别能力,在许多情况中都具有重要意义。

6.2 模式的特征

图像和声音等的模式,可以考虑表示成某种符号。图 6.1(a)所示图形,表示“机器人”,图 6.1(b)是表示日语“我”字的语音波形的例子。人们根据所见所闻,很容易将模式信息变换成“机器人”,“我”等的符号。

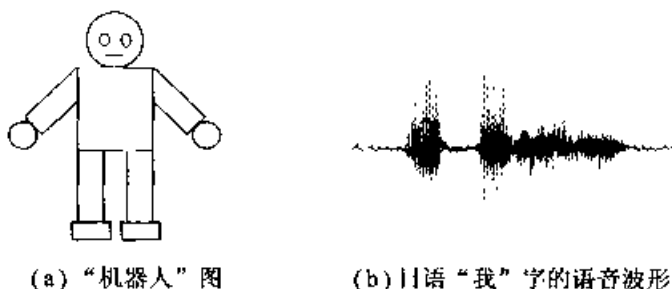


图 6.1 模式的例子

在识别模式时,首先必须知道,根据什么样的特征,可以对模式进行区分。人们通过别人的教诲和从经验中的学习,知道了应该根据什么特征区分“机器人”的图和“汽车”的图,或许还可以知道人类发出的声音与钢琴的声音有什么区别。

在进行模式识别时,应当采用什么样的特征呢? 模式一般具有各种各样的特征,在模式识别时,必须从中选择适当的特征加以应用。作为特征,如果是图形,可以取面积、颜色、边的数目等,如果是声音,则可以取声音的大小、音调的高度、频率分量的强度等,基于设定识别对象的模式,这种选定的特征可以使用,并且其特征会变为不同的量值。此外,即使是相同模式的识别,根据模式识别的目的,可以使用的特征也是不同的,根据使用的特征,模式识别的性能会有很大变化。例如,作为/a/和/i/这两个不同的发音,如果要对其进行识别,那么必须采用频率成分的强度,如果要区分男性的声音和女性的声音,那么采用声音高度的方法是比较好的。因此,模式识别的特征选择,在很大程度上依赖于各个模式识别问题。确定适当的特征的问题,作为一般化问题进行讨论是很困难的,所以我们认为,通过对每个问题的经验和模式的分析等,然后确定使用特征是必要的。

由给定的模式求其特征的处理,称为**特征提取**。得到的特征,

一般用下列特征模式(或特征向量)来表示:

$$x = (x_1, x_2, \dots, x_n)' \quad (6.1)$$

一个模式可以用由特征轴定义的 n 维特征模式空间中的一点来表示。 n 是由一个模式得到的特征数,称为特征模式空间(或特征模式)的维数。

在模式识别中,一般都明确地给出要识别的模式归属于哪一类,大多数被考虑的问题属于这种情况,在把模式识别作为信息处理中的一部分进行考虑时,不仅模式的特征需要确定,也有必要根据目的对类别进行适当的设定。在对图 6.1(b)中表示的声音进行识别时,/w/,/a/,...这样一些音素被分别看作为一类短声音区间进行识别,其后,以这些识别结果作为总体,就可以识别出“我(/watashi/)”这一结果来。另一方面,也可以把“我”作为一类,直接对单词进行识别。同样,对于(a)中图形的例子,如果可以把圆和四方形作为类而构成的部件进行识别的话,那么把“机器人”作为一类进行识别也是可行的。设定成什么样的类别,取决于实施模式识别的简易程度,和对识别结果进行利用的简易程度等。

6.3 根据特征模式匹配进行识别

作为模式识别的一种方法,是先将各类用属于该类的特征模式来表示,然后对输入的特征模式进行判断,看它与哪一类的特征模式相近似。这里,我们称代表类别的特征模式为参考模式(或模板),称输入特征模式为输入模式。

6.3.1 用一个参考模式代表类

作为开始,我们先来考虑输入模式归属于两个类型中的哪一类的识别情况。设输入模式为

$$y = (y_1, y_2, \dots, y_n)' \quad (6.2)$$

两个类型的参考模式分别为

$$r^{(1)} = (r_1^{(1)}, r_2^{(1)}, \dots, r_n^{(1)})' \quad (6.3)$$

$$r^{(2)} = (r_1^{(2)}, r_2^{(2)}, \dots, r_n^{(2)})' \quad (6.4)$$

在这里,测量特征模式相似程度的最基本方法,是利用向量间距离的方法。分别计算输入模式与两个参考模式间的距离,距离较小一方的参考模式的类别就是输入模式被识别出的类别。这种识

别,称之为基于最小距离的识别。输入模式 y 与参考模式 $r^{(c)}$ 的距离 $d(y, r^{(c)})$, 用下列欧几里得距离来定义:

$$d(y, r^{(c)}) = \sqrt{\sum_{i=1}^n (y_i - r_i^{(c)})^2} \quad (6.5)$$

若设输入模式被识别出的类别(识别结果)为 \hat{c} , 则

$$\hat{c} = \begin{cases} 1 & \text{若 } d(y, r^{(1)}) < d(y, r^{(2)}) \\ 2 & \text{若 } d(y, r^{(1)}) > d(y, r^{(2)}) \end{cases} \quad (6.6)$$

在识别中采用的函数,一般称为识别函数。识别函数被定义在每一个类别上,输入模式属于该类时,取比较大的值,属于其他类时,取较小的值。通过应用识别函数 $g^{(c)}(y)$, 基于最短距离的模式识别可以写成下列形式:

$$\hat{c} = \begin{cases} 1 & \text{若 } g^{(1)}(y) > g^{(2)}(y) \\ 2 & \text{若 } g^{(1)}(y) < g^{(2)}(y) \end{cases} \quad (6.7)$$

在式(6.6)中,通过设 $-d(y, r^{(c)})$ 为识别函数 $g^{(c)}(y)$, 式(6.6)变为与式(6.7)相同的形式。在识别函数中,每个的绝对值的大小是没有意义的,但是相对于其他类的大小却成了问题。因此,为了便于进行比较,去掉共同项,将 $d(y, r^{(c)})$ 作平方运算,减去 $\sum_{i=1}^n y_i^2$, 再乘以 $-1/2$, 于是可以设识别函数为

$$g^{(c)}(y) = \sum_{i=1}^n y_i r_i^{(c)} - \frac{1}{2} \sum_{i=1}^n (r_i^{(c)})^2 \quad (6.8)$$

在这种情况下,识别函数变为 y 的一次式,称这种识别函数为线性识别函数。

在这里,让我们来考虑模式空间上的类的边界(识别分界面)。因为边界是由识别函数值相等的点构成的,所以根据式(6.8)得到

$$\sum_{i=1}^n y_i r_i^{(1)} - \frac{1}{2} \sum_{i=1}^n (r_i^{(1)})^2 = \sum_{i=1}^n y_i r_i^{(2)} - \frac{1}{2} \sum_{i=1}^n (r_i^{(2)})^2 \quad (6.9)$$

$$\sum_{i=1}^n (r_i^{(1)} - r_i^{(2)}) y_i = \frac{1}{2} \sum_{i=1}^n (r_i^{(1)2} - r_i^{(2)2}) \quad (6.10)$$

如果用向量表示上式,则得到

$$(r^{(1)} - r^{(2)})^t y = \frac{1}{2} (r^{(1)} - r^{(2)})^t (r^{(1)} + r^{(2)}) \quad (6.11)$$

$$(r^{(1)} - r^{(2)})^t \left(y - \frac{r^{(1)} + r^{(2)}}{2} \right) = 0 \quad (6.12)$$

$(r^{(1)} + r^{(2)})/2$ 是参考模式的中点,该点与输入模式连结的直线为 $y = (r^{(1)} + r^{(2)})/2$,它与连结参考模式的直线 $(r^{(1)} - r^{(2)})$ 呈正交。在特征模式空间为二维的情况中,连结参考模式的直线的垂直二等分线,显然变成了识别边界。一般来说,在识别函数为线性的情况下,识别边界也是线性的,这是众所周知的。

以上我们对类数为 2 的情况进行了阐述,对于 3 类以上的情况,也可以用同样的考虑方法进行模式识别,在这种场合中,将式 (6.7) 设成

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} g^{(c)}(y) \quad (6.13)$$

($\operatorname{argmax}_i x(i)$ 表示使 $x(i)$ 达到最大的 i)。在这里, c 表示所有的类。

此前,参考模式都是作为已经给出的内容进行讨论的,那么参考模式是怎样确定的呢?包含着确定参考模式的模式识别处理流程图,这时变成了图 6.2 那样。正如图 6.2 所示,模式识别的问题可以区分为学习和识别两个过程。学习是为了建造识别系统而进行的一种处理,代表类的参考系统是通过学习确定的,每个类都被存储在系统中。经过这样的学习进行识别就成为可能的了。通常,使用隶属类别清楚的几个模式(例子)进行学习。这样的学习方法被称为有教师的学习。另一方面,例题隶属的正确的类别不知道时,则这种情况下的学习称为无教师学习。无教师学习,在应用识别系统中,必须为达到适用性而逐步更新参考模式等,它是在应用输入模式的识别结果等情况下进行学习的。

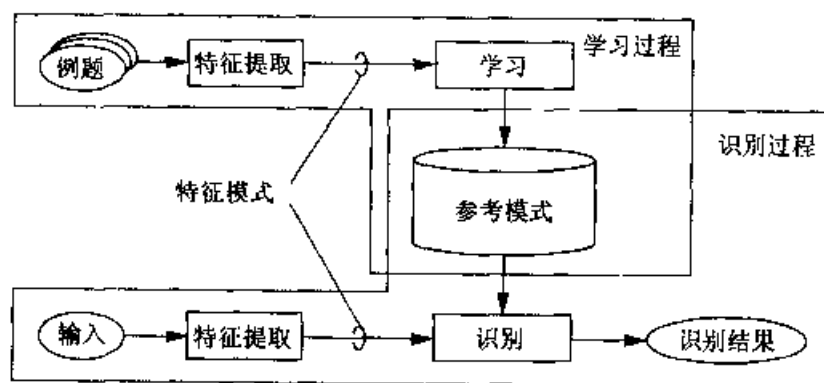


图 6.2 模式识别的处理过程

在用一个参考模式代表类的情况中,在学习中所用到的例题内,求该类的例题的重心,这时若设定参考模式就可以了。对于类 c 的例题,如果设定 $e_k (k=1, \dots, M^c)$ 的特征模式为

$$\mathbf{x}_k = (x_{k1}, x_{k2}, \dots, x_{kn})^T \quad (6.14)$$

则其重心为

$$\mathbf{r}^{(c)} = (r_1^{(c)}, r_2^{(c)}, \dots, r_n^{(c)})^T \quad (6.15)$$

式中

$$r_i^{(c)} = \frac{1}{M} \sum_{k=1}^M x_{ki} \quad (6.16)$$

这就变成了类 c 的参考模式。

6.3.2 用多个参考模式代表类

用一个参考模式表现类的方法,如图 6.3(a)所示,在各类的特征是以某一数据为中心呈圆状(球状)分布的情况下,可以顺利地进行识别,但是当情况不是这样时,例如图 6.3(b)那样的呈阵列型分布的情况下,就不能顺利地进行识别了。这个图表示了特征模式为二维时的例题的分布,○和△表示配置在特征模式空间内的两个类的各自的特征模式,被涂成黑色的点表示参考模式。另外,图中画出的直线,表示两个类的识别边界,就图 6.3(b)而言,对于一些不同的例题,会识别出不同的类别来。

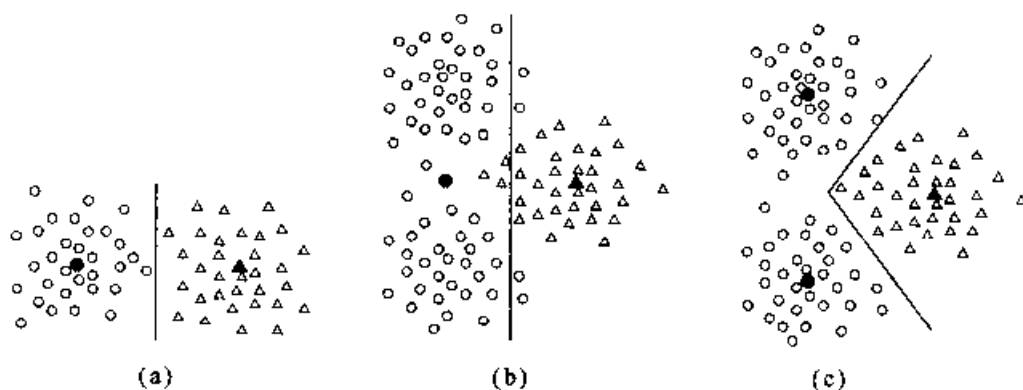


图 6.3 特征模式的分布举例

模式识别的性能评价

在模式识别系统的性能评价中,有在学习用例题进行评价的闭式评价,和在学习不用例题进行评价的开式评价。一般来说,闭式评价的方法比开式评价的识别率更好。学习中用到的例题不充分时,作为对象的类不是本质的,只有学习例题中具有的共同性质,才可以作为参考模式被得到。即

使用闭式评价得出的结论是非常好的,用开式评价也会变得比较差。在学习时,必须注意采用那些只能充分反映各个类的母种群性质的数量的例题。

在进行开式评价时,必须将收集到的例题区分为学习用例题和评价用例题。例如,全部例题区分成两半,分别用作学习和评价。另外,无论是学习还是评价,为了作到尽可能多的利用例题,还存在有被称为交叉证实(cross validation)的评价方法,它将以下列方式进行:将全部例题划分成 L 个(限度为 10)的集,然后用 $L-1$ 个集进行学习,用剩余的一个集进行开式评价,将作为评价用的集依次更换,并且重复进行评价,最后用它们的平均值测定识别系统的性能。

通过对类参考模式的多次关注,可以防止这种误识别发生。根据对类内例题的类均衡处理,由类似的例题可以划分构成几个组,根据由每个组构成的参考模式,可以形成类的多个参考模式。关于类均衡,虽然提出了种种不同的方法,但是在这里只介绍有代表性的类均衡方法,即 k 平均(k -means)法。 k 平均法是一种把例题集 (x_1, x_2, \dots, x_M) , 按照预先制定的数群进行分割的方法,例如,考虑分割成 L 个群 C_1, C_2, \dots, C_L 时的情况,这时可以按以下步骤进行:

第 1 步:从例题集中适当地选取 L 个数据,并且把它们设成各群的中心 $m_l (l=1, \dots, L)$ 。

第 2 步:对所有的例题求距离最小的 m_l , 设其属于群 C_l 。

第 3 步:对于各群 C_l , 求属于它们的例题的中心, 并设其为 m_l' 。

第 4 步:对于所有的群,如果 $m_l = m_l'$ 成立,则结束。否则,则将 m_l' 作为 m_l 并转入第 2 步。

在这种方法中,对于类 c 的 L' 个群,当预置出每个群的参考模式 $r_1^{(c)}, r_2^{(c)}, \dots, r_{L'}^{(c)}$ 后,对于输入模式可以求得

$$d^{(c)} = \min_{i=1, \dots, L'} \|d(y, r_i^{(c)})\| \quad (6.17)$$

但这里假设条件

$$\hat{c} = \operatorname{argmin}_{c \in C} d^{(c)} \quad (6.18)$$

成立。 $(\operatorname{argmin}_i x(i))$ 表示使 $x(i)$ 达到最小的 i 。这种在一个类中,预先选择多个参考模式(模板)的方法,称为多模板法。图 6.3(c)中的例题分布与(b)中的分布是相同的,对于 \bigcirc 类,由于采用了两个模板,所以如图所示边界变成了折线,显然这时被巧妙地分离成了两个类。

另外,用多模板法会作出数量非常多的参考模式,但是把所有的例题考虑为参考模式的识别方法也是存在的。这时把与多模板法相同的输入模式划分为最接近例题的类的方法,称为 NN(nearest neighbor)识别法。此外,代替只根据最近的一个例题进行判断,为从输入模式中寻找最近的 k 个例题,并且,把这些例题中为数最多的例题归属的类作为识别结果,这样的方法被称为 k -NN(k -nearest neighbor)识别法。

在进行上述 NN 识别时,有必要事先对多数的例题进行记忆,并且必须计算输入模式与所有例题的距离。为此,在 NN 识别中,在例题数多的情况下,记忆量和计算量的增大会成为问题。

6.4 基于统计决策理论的识别

下面,在统计决策理论的基础上对模式进行识别。当把属于同一类的模式的集合作为母种群时,我们来确定,考虑应该识别的输入模式是从哪个母种群产生的这种思路是不是最妥善的。这里,当错误的判断了由类 c_i 产生的模式 y 属于类 c_j 时,为考虑其损失,我们用损失函数表示。当设这个损失函数为 $\lambda(c_j/c_i)$ 时,对输入模式 y ,其损失的期望值变为

$$L(c_j | y) = \sum_{i=1}^L \lambda(c_j | c_i) P(c_i | y) \quad (6.19)$$

可以判断,使该值达到最小的 c_i 就是 y 归属的类。这种方法称为给出最小损失的贝叶斯识别。 $P(c_i | y)$ 是所谓的条件概率,它是在知道了被观测的模式为 y 时,类 c_i 的发生概率。进一步讲,当设损失函数为

$$\lambda(c_j | c_i) = \begin{cases} 0 & \text{若 } i=j \\ 1 & \text{若 } i \neq j \end{cases} \quad (6.20)$$

时,称之为后验概率最大贝叶斯识别。

一般来说,式(6.19)的 $P(c_i | y)$ 不能直接计算,因此应用贝叶斯定理将其变换成下列形式:

$$P(c_i | y) = \frac{P(y | c_i) P(c_i)}{P(y)} \quad (6.21)$$

因为 $P(y)$ 与类无关,所以最终在后验概率的贝叶斯识别方面,如果设定

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} P(y|c)P(c) \quad (6.22)$$

则显然问题便可以解决。进一步讲,在类的先验概率相等的情况下,上式变为

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} P(y|c) \quad (6.23)$$

条件概率 $P(y|c)$, 对于类 c 而言, 它是生成特征模式 y 的概率。在对它进行计算时, 对每类都用概率模型表示特征模式是必要的。在实际的模式识别中, 如图 6.3(a) 所示, 其特征模式以某点为中心进行分布的情况是经常见到的。在这种情况下, 根据正态分布, 特征模式的分布可以近似地表示成

$$P(y|c) = \frac{1}{(2\pi)^{n/2} |\Sigma^c|^{1/2}} \exp \left[-\frac{1}{2} (y - m^c)^t \Sigma^{c-1} (y - m^c) \right] \quad (6.24)$$

式中, m^c 和 Σ^c 分别是类 c 的特征模式的均值向量和协方差矩阵, 由学习例题决定。

这样在设定了模式的概率分布后, 利用其参数进行识别的方法称为参数识别, 而在 6.3 节中讲过的那种只利用模式的模式匹配方法, 则称为非参数识别。

6.5 对声音的识别

到 6.4 节为止, 我们说明了模式识别的一般方法。在实际的模式识别问题中, 关于特征模式和类的问题的固有性质被灵活地组合了进来, 因此有必要以适当的方法对其进行识别。本节将对作为模式识别具体问题的应用例子, 即声音识别问题进行说明。首先, 针对单词语音识别问题, 对应用特征模式匹配的方法, 和基于统计决策理论的方法进行说明, 最后再简单地介绍基于统计决策理论的连续声音的识别问题。

声音模式的一个特征, 是其特性可以作为时间序列给出。声音波形的分析, 通常是对图 6.4 中所示的那种被称为帧(或称分析窗)的波形, 在 10~30ms 的短区间内进行的分析, 并且这种分析是沿着时间方向边移动帧边重复进行的。由这种一帧部分的分析, 可以得到一个特征模式, 特征模式的时间序列, 就表示了声音波形全体的特征。特征模式的维数达到 10~200 的程度, 它与采

用的特征的种类密切相关。关于实际中采用的特征,读者可以参考其他一些文献^[1~4]等。

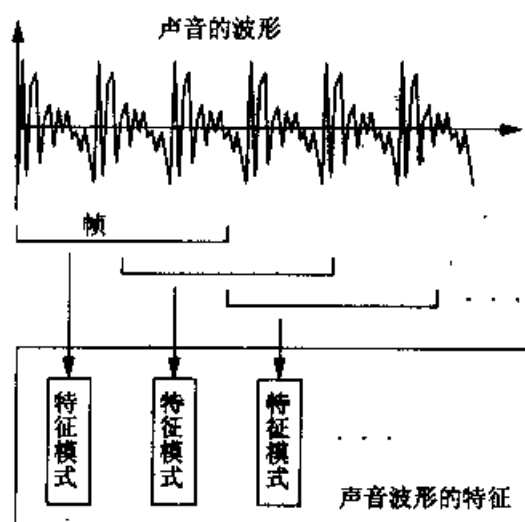


图 6.4 声音分析

6.5.1 根据与参考模式的匹配识别单词

对单独发声的单词语音进行识别,称为孤立单词识别(或者只称单词识别)。在孤立单词识别中,预先把一些单词以某种形式登记到系统中,然后确定说话者说出的单词是其中的哪一个单词。说话者只能就登记中的某一个单词进行发声,这是应遵循的条件。

进行孤立单词识别的一种方法,是在 6.3 节中介绍过的基于与参考模式匹配的方法。但是在声音识别的情况中,作为一个具有代表性方面,用特征模式的时间序列,取代了单一的特征模式(下面对用于每个单词的参考模式序列和未知输入的特征模式序列,分别简称为参考模式和输入模式)。

那么,被登记的单词的参考模式和输入模式之间的距离,怎样进行计算才是合适的呢?在声音模式的匹配中,必须考虑下列问题:

(1) 参考模式与输入模式长度不同的情况下,该怎么办?对于不同的单词和不同的说话者,当然会是这样,即使是同一说话者说同一个单词,也未必构成同样的长度。

(2) 当发音的时间构造发生波动时,怎样进行处理?由于发音的不同或者说话者的不同,单词的某一部分可能说得快一些,也可能说得慢一些。图 6.5 是两个具有一维特征模式,且长度相同的时间序列模式,在模式上的比较图。当把将 A 的时间构造作稍

许变化后的 A' , 与单纯的 A 进行比较时, 就会看出, 两者的特征模式之间可能会有很大的差距。

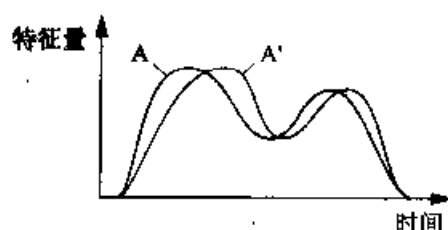


图 6.5 特征模式时间构造的变化

对于问题(1), 将参考模式或输入模式中不管是哪一个, 如进行一样的(与时刻无关, 采用相同的比率)伸缩, 则问题即可解决。在解决问题(2)时, 有必要使时间轴作非线性伸缩。换句话说, 这时有必要通过对基于时刻的模式, 进行伸张或收缩, 使特征模式时间序列的对应关系变得最好, 从而使两个模式达到相互匹配。

使两个特征模式的时间序列作非线性伸缩, 以达到匹配的问题, 可以考虑为图 6.6 中表示的最优路径的搜索问题。当把两个特征模式时间序列 A 和 B (帧数(时间长)分别为 I, J) 分别配置于 x 轴和 y 轴时, 沿着从点 $(1, 1)$ 到点 (I, J) 的方格点形成的各种路径, 应与模式匹配的显示方位逐个地达到一致。特别是连结点 $(1, 1)$ 与点 (I, J) 的直线路径, 它随着时间轴的伸缩进行同样的伸缩, 从而构成了线性收缩的匹配。这里当我们把特征模式时间序列作为声音的特征考虑时, 像图 6.7(a), (b) 那样的沿 x 轴方向, 或沿 y 轴方向的非单调增加的路径(带有返回的路径), 就没有必要去考虑了。在这种形式的路径中, 相应于图 6.7 中的两种特征模式时

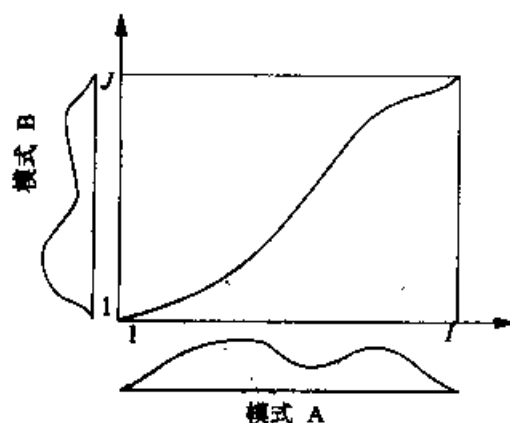


图 6.6 特征模式时间序列的匹配

间序列,在时间轴上构成了部分模式反向运行的现象。排除掉这种类型的路径,对从点(1,1)到点(I,J)的所有路径求模式匹配的距离,若设其最小值为模式间的距离就可以了。这种问题采用动态规划法(dynamic programming)可以获得良好的计算效率。利用动态规划法,使两个特征模式匹配,进而求距离的方法,称为 DP 匹配或 DTW(dynamic time warping)。

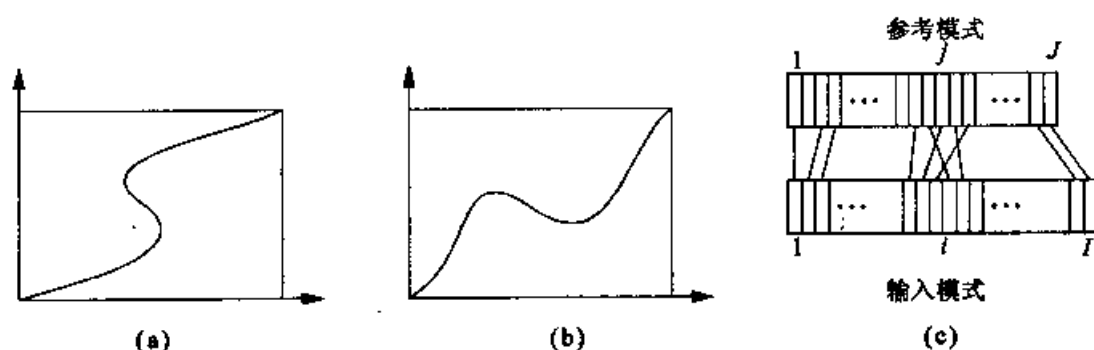


图 6.7 不自然的 DP 路径例子

现在我们设 Y 和 R 分别为输入模式和单词的参考模式,并且设

$$Y = y_1, y_2, \dots, y_I \quad (6.25)$$

$$R = r_1, r_2, \dots, r_J \quad (6.26)$$

I, J 分别是 Y, R 的长度(帧数)。这时,两个模式之间的距离 $D(Y, R)$ 可以用下列 DP 匹配算法求解。

第 1 步: $g(1,1) = d(1,1)$

$$g(1,j) = \infty (j=2, \dots, J)$$

第 2 步: 针对 $i=2, 3, \dots, I$, 实施步骤 3, 步骤 4

第 3 步: 针对 $j=i-r, i-r+1, \dots, i+r$, 实施步骤 4

(式中, 如果 $j < 0$ 则 $j=1$, 如果 $j > J$ 则 $j=J$)

第 4 步:

$$g(i,j) = \min \begin{cases} g(i-1,j) \\ g(i-1,j-1) \\ g(i-1,j-2) \end{cases} + d(i,j) \quad (6.27)$$

第 5 步: $D(Y, R) = g(I, J) / I$

式中, $d(i,j)$ 是 y_i 与 r_j 的向量间距离, 可以根据下列欧几里得距离公式等进行计算。

$$d(i,j) = \|y_i - r_j\| = \sqrt{\sum_{k=1}^n (y_{ik} - r_{jk})^2} \quad (6.28)$$

另外, $g(i, j)$ 表示到方格点 (i, j) 的最小累积距离, 即表示使输入模式的第 i 帧与参考模式的第 j 帧匹配时, 通过最优路径场合下的累积距离。

这里, 让我们来考虑式 (6.27) 的递推公式。该式如图 6.8(a) 中所示, 限制了连接到某方格点的前一个方格点。这种对路径的限制, 对于时间轴的伸缩来说, 不会造成只是使某一部分过分地伸长。作为 DP 路径, 图 6.8(b) 中表示的那种对称形路径也得到了良好应用, 在这种情况下, 步骤 4 的递推公式变为

$$g(i, j) = \min \begin{cases} g(i-1, j) & + d(i, j) \\ g(i-1, j-1) & + 2d(i, j) \\ g(i, j-1) & + d(i, j) \end{cases} \quad (6.29)$$

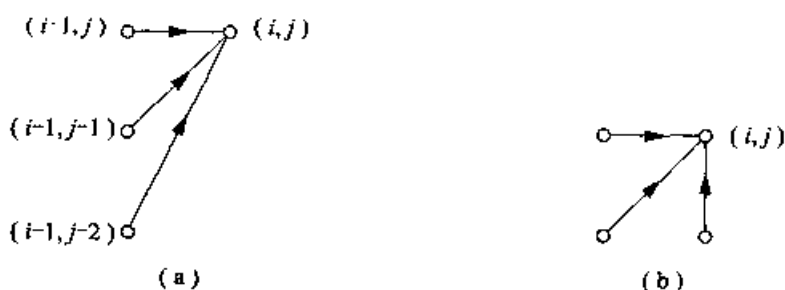


图 6.8 DP 路径举例

这里值得注意的是, 在第二项的路径中, 帧间距离 $d(i, j)$ 的加权增加到原来的两倍。在从始点到终点通过的方格点数, 由于路径的不同可能是有差异的情况下, 为了做到对哪一条路径来说, 通过的方格点数在实际效果上都是相等的, 上述作法就是必要的了。在式 (6.27) 的情况中, 通常在 i 方向上, 每增进 1 不管取哪一条路径, 通过的方格点数都会变为 I 个。在式 (6.29) 的情况中, 由于路径不同, 通过的方格点数会有差异, 对于歪斜的路径, 因为其帧间距离做成了 2 倍, 所以通过的方格点的实际有效数会达到 $I+J$ 。因此, 在这种情况下, 步骤 5 变为

$$D(Y, R) = g(I, J) / (I + J) \quad (6.30)$$

其次, 我们来考虑步骤 3 中的循环范围。为了考察所有的可能性, 虽然有必要设 $j=1, 2, \dots, J$, 但是由于把远远偏离于线性伸缩之外的非线性伸缩排除到了搜索对象以外, 所以可以减小计算量。这样一来, 在图 6.9 中就只对包容在网线以内的路径进行计算了。这

种限制路径的允许区域称为设定匹配窗。在图 6.9 的例子中, B 的路径是被允许的, 但是 A 的路径却被排除到计算对象以外。

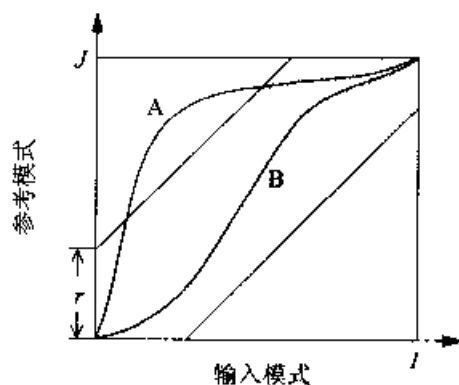


图 6.9 匹配窗举例

6.5.2 基于统计决策理论的单词识别

作为进行声音识别的另一种方法, 是基于统计决策理论的方法。在 6.4 节中, 作为基于统计决策理论的模式识别方法, 表示了用正态分布描述的特征模式分布的例子, 在单词声音的识别中, 有必要用概率模型表示特征模式的时间序列。为此, 应针对每一个单词考虑生成特征模式时间序列的信号源, 并且尝试用概率模型表示这种信号源, 现在作为信号源的模型, HMM(hidden Markov model, 隐马尔可夫模型)得到了广泛应用。图 6.10 模拟地表示了 HMM 的例子。

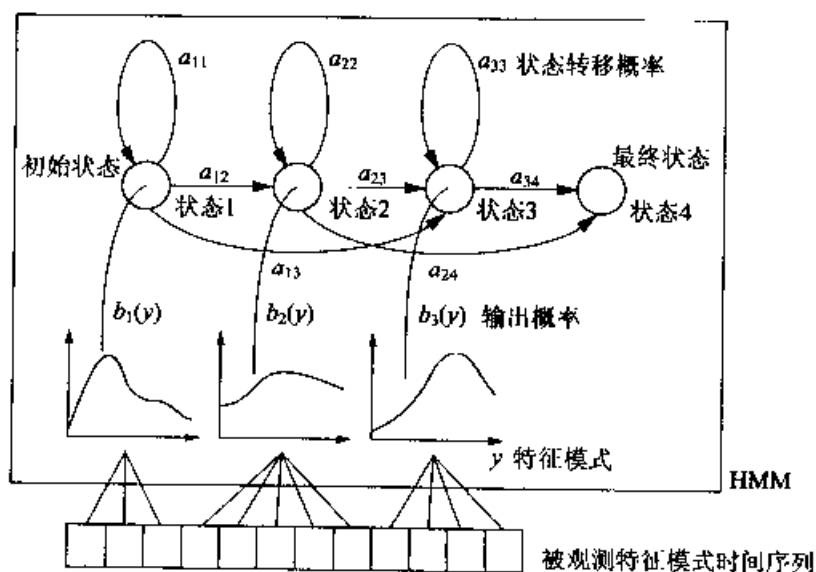


图 6.10 HMM

一般来说, HMM 由状态和状态间的连线决定形状(拓扑结构), 它具有下列参数:

- 状态转移概率 a_{ij} 从状态 i 转移到下一个时刻的状态 j 的概率。 $\sum_j a_{ij} = 1$ 。
- 输出概率 $b_{ij}(y)$ 从状态 i 转移到状态 j 时, 输出特征模式 y 的概率。 $\int b_{ij}(y) dy = 1$ (在图 6.10 的例子中, 输出概率与转移前的状态无关, $b_{11}(y) = b_{12}(y) = \cdots = b_{1l}(y)$)。
- 初始状态概率 π_i 初始状态概率是状态 i 的概率。 $\sum \pi_i = 1$ 。
- 最终状态的集合 F 。

在基于 HMM 的模型化中, 伴随着时刻的推进, 模型内的状态会发生转移(其中包括向相同状态的转移), 根据这种状态转移, 考虑特征模式的输出问题。这里能观测到的只有输出特征模式的时间序列, 各时刻的状态是不能知道的。因此, 称其为“隐”马尔可夫模型。

声音识别的现状

从 20 世纪 80 年代后半期开始, 声音数据库的配备取得了进展, 从而使大量的声音数据的利用成为可能。与此相应, 由于基于统计方法的声音识别也在广泛地进行着, 所以声音识别的性能得到了很大提高。对于数百个孤立的单词, 即使针对非特定的说话者, 其可识别的程度也达到了近 100%。另外, 对于非特定说话者进行认真地大声朗读时连续声音的识别, 当词汇量(可以发声的单词数)在 7000 词时, 单词的识别率(文中被正确识别的单词的比例)超过了 90%。近来, 对于词汇量进一步扩大到 50000 个词的连续声音的识别, 也在试验性进行着, 其单词识别率已达到了 70%~80%。

在这种实验室水平的评价实验方面, 已经获得了相当多的识别性能, 但是在社会上使用的应用系统中, 关于声音识别技术的利用, 还没有取得太多进展。在大词汇量条件下, 不仅要求高识别率, 在实际应用中, 具有鲁棒性的声音识别也是必要的。为了适应周围的噪声环境, 面对着包含有模糊不清的单词和未知单词(未登记到识别系统内的单词)等情况, 并且在发生识别错误时, 实现对话恢复等方面的鲁棒性, 解决这个问题将是一个重大课题。

在利用 HMM 进行声音识别时,有必要考虑下列一些基本问题。

1. HMM 的拓扑决策

在声音识别方面,像图 6.10 中所示,作为初始状态和最终状态,分别各允许一个状态,它们之间依次进行转移,并且恰当地采用了所谓 left-to-right 型拓扑规则。在这种情况下,存在着下列关系:

$$\pi_1 = 1, \pi_2 = \pi_3 = \cdots = 0$$

2. 输出概率的计算方法

输出概率,表示在状态转移时,什么样的特征模式容易被输出,对作为连续量给出的特征模式,在依其原来形式进行处理的情况中,称为**连续 HMM**,在利用向量量化等方法把特征模式变换成符号时,对其符号的时间序列进行处理的情况,称为**离散 HMM**。

在利用连续 HMM 的声音识别中,用 6.4 节介绍过的正态分布或混合正态分布,可以近似表示普通特征模式的分布。在混合正态分布中,把 $b_{ijm}(y)$ 作为以式(6.24)给出的正态分布,则有

$$b_i(y) = \sum_{m=1}^M \lambda_{ijm} b_{ijm}(y) \quad (6.31)$$

这样的公式,用来计算作为 M 个正态分布和的输出概率。式中

$$\lambda_{ijm} \geq 0 \quad (6.32)$$

$$\sum_{m=1}^M \lambda_{ijm} = 1 \quad (6.33)$$

对于离散 HMM,因为 HMM 的输出符号构成了有限集合,所以要预先求出 $b_i(s)$ (s 表示符号),并放进表格中。

3. 模型的评价

根据考虑把状态转移序列 $X = x(0), x(1), x(2), \cdots, x(T)$ 作为允许的所有序列,某个 HMM 的模型 C 输出特征模式时间序列 $y = y_1, y_2, \cdots, y_T$ 的概率,可以用下式计算:

$$P(y | C) = \sum_X \pi_{x(0)} \prod_{t=1}^T a_{x(t-1)x(t)} b_{x(t-1)x(t)}(y_t) \quad (6.34)$$

实际上,多半都以下式进行近似计算:

$$P(y | C) = \max_X \left[\pi_{x(0)} \prod_{t=1}^T a_{x(t-1)x(t)} b_{x(t-1)x(t)}(y_t) \right] \quad (6.35)$$

用式(6.35)进行声音识别的算法称为维塔比(Viterbi)算法。式(6.35)的计算采用动态规划法,下列步骤可以得到良好的计算效果。

第 1 步: $f(i, 0) = \pi_i$

第2步:对于 $t=1, 2, 3, \dots, T$, 执行步骤3和步骤4

第3步:对于所有的状态 i , 执行步骤4

第4步: $f(i, t) = \max_j [f(j, t-1) a_{ij} b_{ij}(y_t)]$ (6.36)

第5步: $P(y|C) = \max_{j \in T} f(j, T)$

4. 模型参数的估计

HMM 的模型, 把转移概率和输出概率作为其模型参数。在表示单词 w 的模型 C 中, 对于 w 的特征模式序列, 为了使 $P(y|C)$ 变大, 必须预先确定模型参数。为此, 利用对单词发声的多种学习数据(特征模式时间序列)决定模型参数。一般来说, 利用基于 EM 算法^[4]的 Baum-Welch 算法等进行这种学习。详细内容请读者参考文献^[5]等资料。

6.5.3 基于统计决策理论的连续声音识别

连续声音识别问题, 是当给出了由 T 帧构成的下列声音特征模式时间序列时:

$$y = y_1 \cdots y_T \quad (6.37)$$

可以考虑找出与 y 适应地最好的单词序列

$$w = w_1 \cdots w_k \quad (6.38)$$

式中, k 为输入中的单词数。通过把声音识别问题固定为上述格式, 就可以求出使 $P(w|y)$ 达到最大的单词序列 w 了。即

$$w = \underset{w}{\operatorname{argmax}} P(w|y) \quad (6.39)$$

根据贝叶斯定理, 可以直接写成

$$P(w|y) = \frac{P(y|w)P(w)}{P(y)} \quad (6.40)$$

$P(y)$ 是声音模式 y 被观测到的先验概率, 因为它与 w 无关, 所以使 $P(y|w)P(w)$ 最大化就可以了。

这里, $P(y|w)$ 是考虑在对单词序列发声时, 声音模式 y 被观测到的概率, 由音响模型给出。另外, $P(w)$ 是单词序列产生的先验概率, 由语言模型给出。基于这种固定化格式, 对于作为输入考虑的所有单词序列(文章), 计算 $P(y|w)P(w)$, 给出最大值的 w 就设定为识别结果。实际上, 为了提高计算效率, 提出了各种各样的办法^[6]。下面就音响模型和语言模型作一些简单介绍。

近年来在连续声音识别中, 作为音响模型, 广泛地采用了 HMM。在 6.5.2 节中介绍的孤立单词识别中, 采用了针对每个单词的 HMM 模型, 在连续声音识别中, 为了控制模型的数量, 通过

对用比较小的/a/, /k/等音素作单位的 HMM 模型的学习予以实现。在进行识别时,由于把模型的最终状态设为下一个模型的初始状态,所以就构成了表示单词和文章的模型,计算该模型生成输入声音模式 y 的概率。图 6.11 是模型的连接例子,它表示了由/a/, /k/, /i/的模型,构成“红的(/akai/)¹⁾”的模型的情况。这样连结成的模型,在对生成输入特征模式的概率进行计算时,只是模型的状态数变多了,因此,基本上按照 6.5.2 节中介绍的要领进行就可以了。

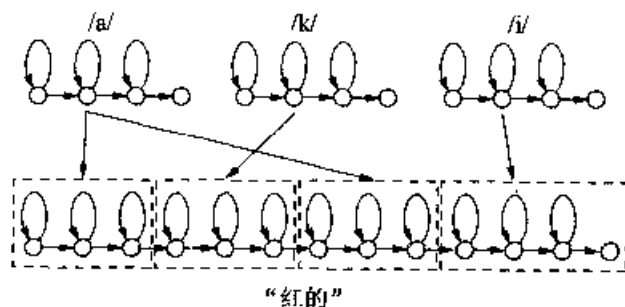


图 6.11 HMM 模型的连结

另一方面,在语言模型中,虽然有必要计算

$$P(w) = P(w_1)P(w_2 | w_1) \cdots P(w_k | w_1 w_2 \cdots w_{k-1}) \quad (6.41)$$

但是因为先于每个单词考虑所有单词的影响是一件困难的事情,所以只考虑与每个单词的发生有关的过去的 $(n-1)$ 个单词的影响,并且近似地表示为

$$\begin{aligned} P(w) &\approx P(w_1 | \cdot) P(w_2 | w_1) \cdots P(w_k | w_{k-(n-1)} \cdots w_{k-1}) \\ &= \sum_{i=1}^k P(w_i | w_{i-(n-1)} \cdots w_{i-1}) \end{aligned} \quad (6.42)$$

这种模型化称为 n -gram, $n=1, 2, 3$ 时的模型,分别称为 unigram, bigram 和 trigram。这种单词的连锁式出现的概率,需应用大量的语言汇编来估计, n 越大,则在汇编中出现的频率就越小,或者说不出现的单词连锁会增加,稳定地估计发生概率是很困难的事情。为此,通常由汇编得到的出现概率用其原来的值来代替,并且采用比较低阶的 n -gram 上的估计结果,对发生概率应用平滑处理的方法^[7]。

1) 日语“红的”的读音为/akai/-----译者注。

练习题

1. 对于特征模式为二维,类数为 2 的模式识别,当给出参考模式为 $r^{(1)} = (2, 5)$, $r^{(2)} = (6, 1)$ 时,试求识别边界会是什么样。
2. 对于具有一维特征模式 x 的例题模式,图 6.12(a)是针对两类(c_1 和 c_2)把 x 的分布作为直方图表示的频率分布图。图 6.1(b)是用正态分布对其进行近似表示的分布图,对于图 6.12(a)中的两种分布,它们各自的均值(μ)和方差(σ^2),分别为 $\mu_1 = 5.0, \sigma_1^2 = 1.0, \mu_2 = 15.0, \sigma_2^2 = 10.0$ 。这时,试针对输入模式 $x = 9.0$,分别计算各类的 $P(x|C_i)$ 。另外,它能被识别成哪一类呢?

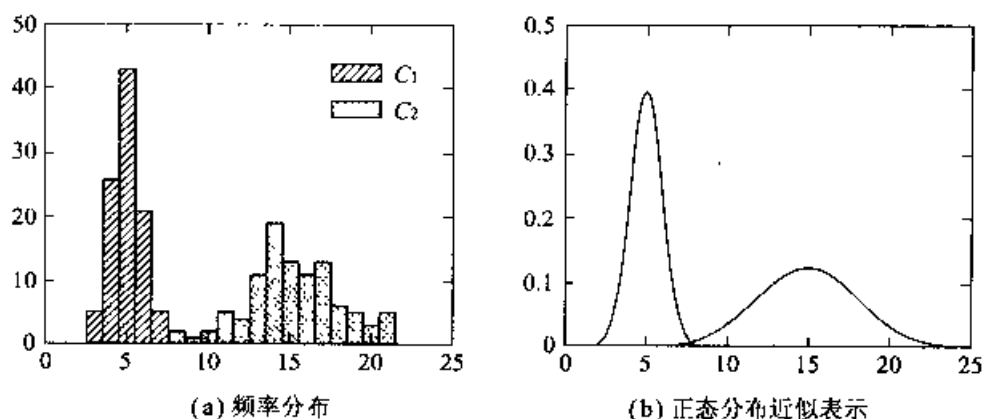


图 6.12 模式的分布例子

3. 基于贝叶斯识别法进行两类的模式识别时,试考虑识别边界会变成什么样的面。这里,设模式分布可以用式(6.24)那样的正态分布近似表示,并假设两类中的协方差矩阵和发生概率是分别相等的。
4. 当特征模式为一维的,输入模式为 $y = (3, 5, 6, 4, 2, 5)$,参考模式为 $r = (5, 6, 3, 1, 5, 6)$ 时,采用图 6.8(a)所示的非对称路径进行 DP 匹配,求模式间的距离。这里,设向量间的距离 $d(i, j) = |y_i - r_j|$ 。另外,求线性伸缩时的距离。
5. 在图 6.13 所示的 left-to-right 型的离散 HMM 中,求输出符号序列 abb

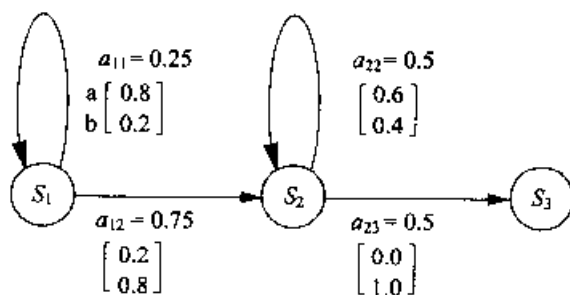


图 6.13 离散 HMM 的例子

的概率。图中,设 S_1 为初始状态, S_4 为最终状态, a_{ij} 为从状态 i 到状态 j 的转移概率,设 $\begin{bmatrix} a & b \end{bmatrix}$ 内表示的是符号 a 和 b 的输出概率(上排为 a 的输出概率)。另外,若用 Viterbi 算法求这个概率,试问会有哪些变化。

第 7 章

人工智能语言

本章将对 Lisp 和 Prolog 这两种语言进行介绍。首先,我们从程序设计语言的观点出发,阐述关于人类知识的特征问题;其次,我们将说明 Lisp 和 Prolog 是怎样表示这些特征的;最后,我们还要介绍一些简单的 Lisp 和 Prolog 的程序例子。本章的重点内容是递归性及程序与数据间的相互变换。这两方面内容的意义将在文中介绍。

7.1 人工智能语言是怎样一种语言

这里所说的人工智能语言,是为开发人工智能系统而采用的程序设计语言。即使是所谓人工智能语言,也并不是什么具有“魔法”的程序设计语言,说到底,它与现在广泛使用的 C++ 和 Java (下面称这种语言为现有型语言)等没有本质的区别¹⁾。但是人工智能语言在记述人工智能系统时,具有应用方便的特征。那么这些特征到底是什么呢?在考虑这些内容时,有必要考虑人工智能研究对象的特征,以及这时的人工设计语言任务。

人工智能是研究在计算机上实现各种人类智能活动的技术,这时,程序设计语言的任务就是为了在计算机上实现研究人员的设计思想,对计算机发出的指示进行记述。因此,人工智能语言只要能真实而简单地表示出研究人员的设计思想,也就达到了人们的期望。人工智能的研究对象变成了人类的智能。因此在研究人工智能时,有必要用某种模型来表示人类的智能。如果是这样,人类的智能又会有怎样的特征呢?在认知科学或心理学等领域,虽然有各种各样的特征被人们倡导,但是在这里,我们从程序设计语

1) 因为不论是现有型语言,还是人工智能语言,都是在相同的 CPU 上进行工作的,所以按理说不应有本质上的差别。

言这种观点出发,作为人类智能的特征,我们列举了“符号性、阶层性、递归性、成长性”这四种特征。¹⁾

首先,所谓“符号性”,意思是“人类利用符号进行各种智能活动”。人类的许多知识,是由符号构成的。人类学习用文字(符号)书写的知识,为学习的知识加上名称(符号)进行记忆,并且根据需要,解决引伸出这些知识的问题。因此,也可以说,人类的许多智能活动都是利用符号进行的。

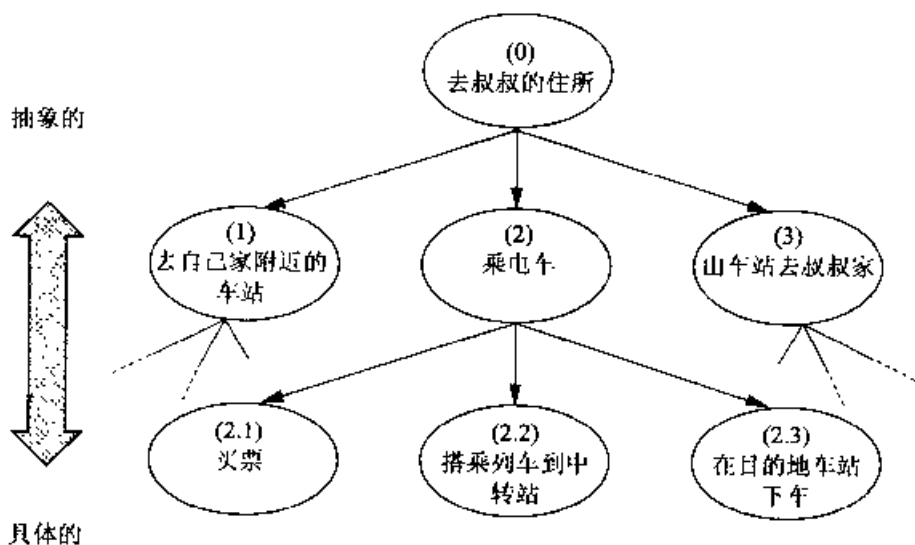


图 7.1 知识的阶层性

计算机,顾名思义,它是一种为把原来的数值作为对象进行计算而开发出来的机械,但是不久,它便做到了不仅能处理数值,还能处理文字等符号,因此,它的应用范围日益广泛。但是在现有型语言中,始终是以“数值”表示的数据处理为主,以符号数据的处理为辅的。可以说“符号也可以处理”这种状况是理所当然的。可是,在人工智能系统中,作为处理对象的知识的大部分内容,最终都是要用符号表示的。这时,时至今日的主从关系就要发生逆转,符号处理占据主导地位的程序设计语言,就成为必要的了。

所谓的“阶层性”,就是“知识被分割成层次”这样一种事实。例如考虑“从自己的住宅去叔叔家”这样一件事。如果是对人类,只要发出请求“请去叔叔家吧”,就可以了,但是对于计算机就不会那么简单,这时必须详细地为计算机教授步骤,这种步骤例如会变

1) 当然,除此之外也还提出了一些特征。详细内容可以参考相关章节的参考文献。

成下列情况：

- (1) 从自己家出来去最近的车站；
- (2) 由车站登上列车，并且在距叔叔家最近的车站下车；
- (3) 从距叔叔家最近的车站走到叔叔家。

进一步讲，例如在(2)这一步上，还可以分割成下列步骤：

- (2.1) 购买去目的车站的车票；
- (2.2) 登上列车；
- (2.3) 在目的车站下车。

在这种分割中，(1)和(3)都是可能的，但是(2.1)等还可以进一步分割成“从钱包里往外掏钱→把钱放进车票销售机→按压费用按钮”。若把这种关系表示于图中，则可以用图 7.1 所示的关系图表示。图 7.1 的上方书写的知识比较抽象，图 7.1 的下方，书写的知识比较具体。这样就可以把知识分成不同层次表示出来，换句话说，当进行分层表示时，就可以把人类的知识更好地表示出来，这就是“智能的阶层性”的意义。

其次，“递归性”这个概念有点不好理解。“递归”这个词本身，也许不是一个经常听到的词。根据某辞典的定义，它被记述为“按照处理程序和规则的定义，其自身被反复使用的一种方法”。可是，递归性的最本质的部分，还是所谓“部分与全体是相似形”这一点。

这里我们要再次观察图 7.1，在这个图中，例如当我们观看(2)(这种位置称为节点，或称为结点)时，由该节点向下方进行引线(这些线称为弧线)，在这些弧线的下端，再次与节点相连。另一方面，当观察(0)这个节点时，同样在它的下方可以引出弧线，而在弧线的前端连上节点。总之，当我们注意到下列情况，即“从节点引伸出弧线，然后再使弧线的前端连接上别的节点”时，我们会发现，节点(0)和(2)具有完全相同的构造。因此，从(2)开始的部分，显然是从(0)开始的部分中的一部分。这种情况就是所谓的“部分与全体是相似形”的概念。递归性的另一个明显的例子是文章语句的结构。例如，考虑下列句子：

I think that you have a book.

根据英语文法，这个句子的结构可以图解为图 7.2 那样。这里值得关注的是 that 以下的部分。这个文章作为全体，具有〈主语(I)〉〈谓语(think)〉〈宾语(that 以下部分)〉这种 S+V+O 的形式，作为这个句子的一部分的 that 以下部分，也同样具有 S+V+O 这

种形式。这也表示了“部分与全体是相似形”这种特征。

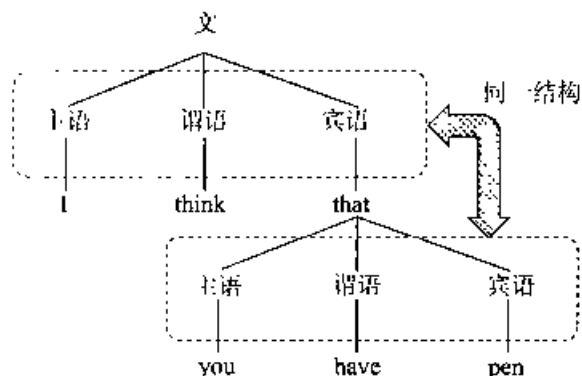


图 7.2 英文的递归结构

前面的“去叔叔家访问”的问题,和后面的“英语句子分析”的例子等,是在规划和自然语言处理这种人工智能系统的极重要的应用领域中,最初步的例子。在这种人类的智能系统中,“部分与全体是相似形”这种情况,出现的频率非常高。

最后,关于“成长性”,简单地说就是“人类的智能通过学习而在不断变化着”。这种“成长”并不是仅进行“稍许努力”就能够看得出来的。在我们的日常生活中,不管我们是否意识到了,实际上每天的智能状况都在变化着。这种“智能的变化”,是人类智能的一大特征。例如,若设今早在那趟车到站的 13 分钟前从家里出来,但却因迟到了少许时间而未能乘上那趟公共汽车,于是我们就获得了下列知识,即“为了能赶上那趟公共汽车,必须在 15 分钟前从家里出来”。这样一来,人类的智能就会不断地完善,从而使情况变得越来越好。

为了能更好地表示上述人类智能的特征,作为程序设计语言,我们希望具备下列一些特征:

- 把符号的操作作为处理的基本内容;
- 对于阶层结构和递归结构等的复杂数据形式,能够简单地进行表示;
- 伴随着系统的运行,系统自身的变化(进化)能够表示出来。

为了实现这些特征,在人工智能语言方面,采用了下列两种方式:

- (1) 把具有递归性质的数据结构,作为数据的基本要素;
- (2) 缓解程序与数据之间的区别。

作为(1)中的“具有递归性质的数据结构”,在 Lisp 中以“表”,在 Prolog 中以“项”这种形式,作为基本要素。(2)具有下列意义。一般对应于状态的变化而变化的东西是数据。因此,如果数据和程序的区别被缓解,能把正在运行的程序作为数据来操作,把操作的数据作为程序来执行,那么就可以动态地改变作为执行主体的程序。因此,由于这种“使区别得到缓解”,使得对系统进化的表示成为可能。

作为具备上述特征的人工智能语言,下面我们列举出函数型语言 Lisp 和逻辑型语言 Prolog 这两种语言。

7.2 函数型语言 Lisp

Lisp 语言是人工智能语言中具有代表性的语言。Lisp 是 20 世纪 50 年代由美国的麦卡锡(J. McCarthy)开发出的一种语言,由于它具备便于在人工智能系统的开发中应用这一特征,所以从应用于初期的人工智能系统的描述以来,现在已经变成了人工智能研究的核心语言。下面我们试分析 Lisp 对 7.1 节中叙述的特征是怎样实现的。

7.2.1 表:具有递归结构的数据

“表”这个词本身并不是什么新颖的词。即使在日常生活中,像“购物表”啦,“俱乐部成员表”啦这样一些情况中,都是在使用上极其普通的词。可是在计算机科学的领域内,如果谈到表,则意味着是某种特定的数据结构。

首先,对原子这种数据结构的定义进行说明、原子这种数据结构最初构成了表的基础。

“所谓原子就是一个一个的完整的文字排列(文字列)”。

因此,原子表示了把文字进行排列后,能够构成的一切内容。例如,像“computer”啦,“田中一郎”啦,或者“1.5”这样一些排列的文字列,全都是“原子”¹⁾。以这种“原子”为基础作成表。于是构成了关于表的下列定义:

“所谓表,就是把原子或表排列起来并且用括号将其围起构成的”。

1) 实际上在程序中记述表时,不要这个“ ”(双引号)。

总之,所谓表就是把原子排列起来,首先构成基本内容。例如,把“computer”和“science”这两个原子排列起来时,就生成了“(computer science)”这个表。

作为表这种数据结构的重要之处,就在于可以作为表的组成元素包含表。最终,可以作出下列形式的表:

(department of (computer science))

这个表由“department”,“of”这两个原子,和“(computer science)”这一个表共计三个元素构成。这个表的数据结构的特征体现为下面两点:

- (1) 构成表的最小元素是原子这样的文字列;
- (2) 在表的元素中,可以包含表(即其本身)。

由特征(1)可以清楚地得知,Lisp 适合于进行符号处理。另外,特征(2)是递归结构本身。表的元素,最终会有部分也变成表,这种情况就变成了前面介绍的“部分与全体是相似形”情况的典型例子。基于这种情况,人工智能程序作为对象,可以表示具有阶层性和递归性的复杂数据结构。

7.2.2 Lisp 程序的基本结构

Lisp 被称为函数型语言,Lisp 的程序变成了函数的集合。换句话说,所谓在 Lisp 上去编写程序,就是按照 Lisp 程序设计语言的语法规则去定义函数。在 Lisp 上进行复杂的处理时,可以通过把简单函数组合在一起来实现。这种成功的考虑方法,与现有型语言的子程序大致是相同的。

在 Lisp 中,一般用前缀表示法(prefix notation)这种形式记述有关函数的表示。所谓前缀表示法,就是把表示函数算子的原子记述在表的第一元素内,把自变量记述在第二元素以下的方法。在 Lisp 中,函数的记述可以写成下列形式:

((函数名)<第一自变量><第二自变量>...)

因此,例如“3+4”表示为(+3 4),“sin(x)”表示成(sin x)这种形式。这样在采用前缀表示法时,就使函数的表示方法具有了一致性。

在我们常用的函数表示法中,Lisp 的那种前缀表示法,与“3+4”这样的算子位于演算对象之间的中缀表示法(infix notation),或者像“5!”那样算子位于演算对象值的后方的后缀表示法(postfix notation),是混杂在一起的。但是,从作为程序设计语言观察

到的情况,或者从构成处理程序设计语言的处理系统的立场考虑,这种混杂的表示方法是不好的。因此,Lisp 这种能够对所有的演算进行统一表示的前缀表示法,被人们所采用。

其次,我们来表示 Lisp 函数的定义,换句话说来表示程序的书写方法。在 Lisp 中定义函数时,仍然采用函数。用到函数定义中的函数,是 defun 这种函数¹⁾:

(defun<函数名><自变量表><函数定义的主体>)

仔细观察上述函数,就可以清楚函数的定义,并且最终可以明白由程序形成的表。总之,在由四个元素构成的表中,第一个元素是 defun 这个原子²⁾,第二个元素是表示函数名的原子,第三个元素是表示自变量的表,第四个元素是表示函数主体(计算的实体)的表。

这里我们举一个 Lisp 程序的例子。这个程序是一个从摄氏温度向华氏温度变换的程序。从摄氏向华氏变换的公式如下:

华氏 = 摄氏 * 9/5 + 32

若用 Lisp 书写上式,则变成下列形式:

```
(defun ctof (ctemp)
  (+ (/ (* ctemp 9) 5) 32))
```

这个程序具有下列意义。程序的名字是 ctof,自变量是一个 ctemp,此外,计算函数值的定义本身为(+ (/ (* ctemp 9) 5) 32)。

那么,让我们再一次看看刚才给出的温度变换程序。这个程序的函数主体部分表示成了“摄氏 * 9/5 + 32”这种形式。这里我们着眼于程序的部分内容。例如(/ (* ctemp 9) 5)表示了“摄氏 * 9/5”这部分计算。这部分计算的意义是“摄氏的值先乘以 9,然后再用 5 去除其值”,当着眼于除法运算部分时,那么这个除法运算的第一自变量,这时就变成了“(* ctemp 9)”这种 Lisp 程序。因此,将上述结果用 5 除,就得到了所需结果。

总之,在 Lisp 的程序(=函数)中,作为函数的自变量,可以给出别的函数。其原因是因为这种函数具有恢复到其一定值的性质。因此,在计算某个函数 f 的值时,先利用另一个函数 g 计算其

1) 严格地讲,defun 不是函数,而是一个宏指令。但是在现在的情况下,把它考虑成函数是可以的。

2) 这在 Lisp 的处理系统中有微妙的区别。在某些处理系统中,用 DE 这种原子代替 defun。另外,也有使用 define 这种原子的处理系统。

自变量 a , 并且将 g 作为 f 的自变量给出时, 就可以在函数 f 的自变量中, 直接地记入调用函数 g 。当这样进行计算时, 在 Lisp 的处理系统中, 就要先对作为自变量的函数 g 的值进行计算, 然后再对整个函数 f 的值进行计算。

在这种 Lisp 语言中, 为了计算作为程序语言基础的函数的自变量, 需要调用“转包”函数, 而为了计算这种“转包”函数, 又需调用“二次转包”函数, 依次进行, 尚需调用再下一次转包函数……, 这样, 在执行程序时, 就必须重叠地进行函数的调用过程。不过这不仅对 Lisp 是这种情况, 同样的情况比如说对 C 语言也会产生, 例如:

$$x = \text{func1}(a + b, \text{func2}(1))$$

该式的含义是“计算把 $a + b$ 作为第一自变量, 把 $\text{func2}(1)$ 作为第二自变量的函数 func1 的值, 并把计算结果代进 x ”, 这里先计算作为第一自变量的 $a + b$, 然后(哪里被定义在其他部分)计算 $\text{func2}(1)$ 的值, 由此计算 func1 的值。这点也和 Lisp 的情况一样, 它们具有相同的运行机构。不过, 如果是在 C 语言的情况下, 采用将函数值代进别的变量等方法, 但在 Lisp 的情况中, 原则上除使用函数的恢复值以外, 不使用其他方法。因此, Lisp 的程序采取函数的自变量也变成函数的那种结构, 而且因为各函数用表来表示, 所以它趋向于使括号稍许减少。

至今在我们看到的 Lisp 中, 因为“程序和数据都是以同一形式表示的”, 所以只看外观方面, 程序也好, 数据也好, 是没有区别的。称表和构成表的更小的单位——原子为 symbolic expression, 简称为 S 式, 在 Lisp 中, 不管是程序还是数据, 哪一个都是用 S 式表示的。例如当我们考虑 $(a\ b\ c)$ 这个表时, 可以作下列两种解释:

- (1) 元素的顺序为 a, b, c 的表(作为数据的表);
- (2) 调出函数名为 a , 自变量为 b 和 c 的函数(作为程序的表)。

只根据形式(外观)是不能判定 $(a\ b\ c)$ 这个文字列是属于哪一种解释的。这是与程序和数据能被明确区分开来的现有型语言的最大区别。

7.2.3 由程序到数据和由数据到程序: eval 和 quote

在 7.1 节说明了“人工智能语言一方面可以把程序作为数据处理, 同时还必须能够把数据作为程序去执行”。在 Lisp 中实现

上述功能的“装置”是 eval 和 quote, 本节将对它们进行说明。

考虑前面的(a b c)表。这里单凭形式是不能用来区分它是程序还是数据的, 这点前面已经谈到, 例如, 我们设它是数据。若作这种假设, 则这个表就变成了第一个元素为原子 a, 第二个元素为原子 b, 第三个元素为原子 c 的表。不过当使用 eval 这个函数时, 就可以把它作为程序来执行了。

eval 这个函数, 只要选取表形式的自变量, 那么它就可以把作为自变量给出的表作为程序, 进行试探性计算。因此, 当设 (eval '(a b c)) 时, 就是设定函数 eval 把自变量的 (a b c) 作为程序去执行。总之, (eval '(a b c)) 的意义, 变成了“去执行把 b 和 c 作为自变量的函数 a”。当然, 在程序中的何处对 a 这个函数进行具体计算的方法, 必须定义出来, 但如果它被恰当地定义了出来, 那么作为数据的 (a b c) 这个表, 就能够作为程序来执行了。

因为变成函数 eval 的自变量的表仅仅是数据, 所以在后面介绍的表被用来作为操作函数, 它可以在执行程序的过程中构成。无论如何, 只要构成了表的形式, 根据设定的函数 eval 自变量, 就可以把它作为程序来执行。

与这个 eval 比较, 具有完全相反作用的函数是 quote。eval 是用来实现“把数据程序化”的函数, 而 quote 则是用来实现“把程序数据化”的函数。例如, (first '(a b c)) 这种文字列, 就是 Lisp 的程序¹⁾。可是, 有时可把它看作是一个表。该表的“第一个元素是 first, 第二个元素是 '(a b c), 总之是数据”的情况。在这种情况下, 正如 (quote (first '(a b c))) 那样, 当把看作为数据那样的程序赋予自变量, 并执行 quote 这种函数时, quote 这种函数就把给定的程序变换成了数据。这个 quote 函数的返回值 (函数一定有返回值) 虽然与原来的程序具有完全相同的文字列, 但是它已经不是程序而变成了数据。因为这个 quote 函数在 Lisp 的程序中频繁使用, 所以存在有使用引号的简略表示方法。'(a b c) 与 (quote (a b c)) 具有相同的意义。在 Lisp 的处理系统中, 当发现 '(a b c) 这种记述时, 会自动地变换成 (quote (a b c)) 这种记述。

由于使用了 eval 和 quote, 所以在 Lisp 中执行程序时, 可以相当自由地进行变更。在判读某个程序的过程中, 应用 quote 函数可以使其数据化, 应用表操作函数可以对其实施变更, 由于应用

1) first 是 Lisp 的一种内部函数。

eval 函数可以对其结果实现循环操作,所以可以实现程序的动态变更。

在用 Lisp 开发人工智能系统的情况中,这种 eval 和 quote 起到了非常重要的作用。进一步讲,如利用这一性质,就能够比较容易地实现构成更适合于自己目的的 Lisp 处理系统。因为详细地讨论超出了本书的范围,所以这里予以省略。当只涉及结论时,则可以用 Lisp,最终用用户程序,对 Lisp 的解释程序(处理系统)进行记述¹⁾。这也是源于 Lisp 特征的一个优点,它使得数据和程序之间的变换能够比较容易地进行²⁾。

7.2.4 表操作

在 Lisp 中,因为所有的数据都是用 S 式表示的,所以表的操作会频繁地进行。因此,在 Lisp 中,对表操作定义了一些方便的功能(函数)。以下列举其中的一些基本功能:

1. 从表中取出元素

表操作的最基本内容,是从表中取出元素。在 Lisp 中,提供了下列两种基本操作:

- 取出表中的最初的元素 这是一种从表中取出以自变量给出的最初元素的操作。由于具有“表的最初元素”这种意义,所以称这个函数为 first³⁾。总之,若设(first '(a b c)),则作为(a b c)这个表的最初元素 a,被得到了。

- 取出表中最初元素以外的元素 从表中取出元素的另一项操作,是“除了最初元素以外,取出剩余的表”这种操作。这种操作被称为 rest 这个名字⁴⁾。若设(rest '(a b c))时,则会得到除最初的元素 a 外,剩余的表(b c)这种表。这里必须注意,由 rest 得到的解答必定是表。例如当取(a b)这个表的 rest 时,其解答不是 b,而得到的是(b)这个表。

这里的 first 和 rest 是从表中取出元素的基本操作。如果存

1) 例如,可以自己写出 C 语言的处理系统,并且希望与之进行比较。

2) 可是,为此在 Lisp 中会产生许多“方言”,即随之产生了弊害。

3) 根据历史的演变,以前把与 first 相当的函数称为 car。市场上出售的 Lisp 的文献书籍也几乎都采用了 car 这个词。这个词是 contents of address register 的缩略词。不过现在 first 这个词已经逐渐渗透到各个领域。但是为了有效地利用以前开发出来的庞大的程序资源,在新的 Lisp 的处理系统中,使用 car 这个函数的情况也很多。

4) 这个 rest 与 first 相同,以前被称为 cdr(contents of data register)。

在这两种操作,那么就能够从表中取出任意的元素。例如,如果想取出表中的第二号元素,那么把 rest 施加到对象的表上,并且从得到的表中取出 first 就可以了。其原因是“原来表中的第二项”,就是“除去最初元素后形成的表的最初元素”。以同样方法进行,取出表的最后的元素的操作,就变成了“取出表的剩余元素时,变成成为空表,也就是变成没有任何东西的表,而只有最初的元素”。

因此,这两种表的操作是最基本的操作,根据对这两种操作的组合,可以取出任何元素。这种处理的形成,使得表这种数据结构具有了递归的性质,而最终即使取表的一部分(在目前情况下,是除去最初元素后剩余元素的表),它也具有变成为表的这种性质。

2. 表的组成

与表的元素取出操作同样,表操作中的另一种最基本的操作,这就是表的组成。在 Lisp 中,通常备有下列三种函数:

- 表的构成 通常称之为 cons。它是 construction 的缩写,采用两个自变量。在 cons 作成的表中,最初的自变量是 first,第二个自变量为 rest。例如 $(\text{cons } '(a\ b) \ '(x\ y))$ 可以返还为 $((a\ b)\ x\ y)$ 这种表的形式。

- 表的连结 通常用 append 这种名称。给定自变量(这里也是两个)的表,一旦做的零乱时,append 就能把零乱的各自变量元素,由左方开始顺次进行排列作成为表。例如 $(\text{append } '(a\ b) \ '(x\ y))$,这时返回到 $(a\ b\ x\ y)$ 这种形式的表。

- 表化 通常称之为 list。list 取任意个自变量,这时会把这些自变量各个都设为元素,返回成表的形式。例如, $(\text{list } '(a\ b) \ '(x\ y))$ 这时变成为 $((a\ b)\ (x\ y))$ 。

在图 7.3 中,图解表示了 cons, append 和 list 这三种函数的不

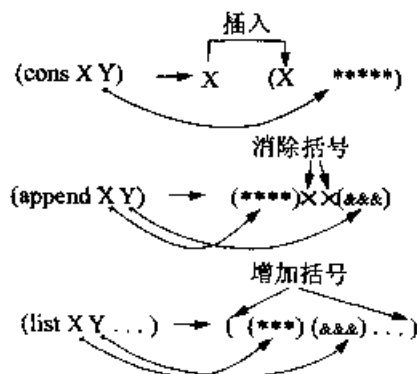


图 7.3 cons, append 和 list 的情况

同功能。希望读者根据这个图解,深入地理解各个关系中自变量与结果的表的关系。

下面介绍 Lisp 的实施控制功能。所谓实施控制,就是条件转移和重复等,对程序的执行顺序进行控制的功能。

1. 条件转移

所谓条件转移,就是“在某个条件 D 成立的情况下,执行程序的 A 部分,在条件 P 不成立的情况下,执行程序的 B 部分”。在 C++ 等现有型语言中,称为 if 语句。在 Lisp 中,条件转移也用表记述,并且可以表示成下列形式:

```
(cond
  (<条件式 1><执行语句 1>)
  (<条件式 2><执行语句 2>)
  :
  (<条件式 n><执行语句 n>))
```

这也变成了表的结构。总之,作为表的全体,这时变成为“第一个元素是称为 cond 的原子,第二个元素是称为(<条件式 1><执行语句 1>)的表, ..., 第 n 个元素是称为(<条件式 n><执行语句 n>)的表,最后由它们共同组成表”。这种表示的意义基本上是明确的,即这意味着“<条件式 1>成立时,执行<执行语句 1>, ..., <条件式 n>成立时,执行<执行语句 n>”。因为这个 cond 语句也是函数的一种,所以可以返回到作为执行结果的值。cond 语句的值,是在条件成立时,被执行的执行语句的执行结果(值)变成的 cond 语句的执行结果(值)。

下面的例子是 cond 语句的例子。

```
(cond ((eq num 1)(print 'one))
      ((eq num 2)(print 'two))
      (t          (print 'too many!!)))
```

这个程序是这样一种程序,这就是如果数据(num)为 1 则打印 one,如果是 2 则打印 two,如果是 2 以上则打印 too many!! 在最后一行中,条件变成了 t 这个文字,它表示“在上述条件全不成立时”¹⁾。总之,它相当于 C 语言等中的 default。

1) t 表示 true,即表示“真”。因此这个条件通常是成立的。

在原来的 Lisp 中,条件转移的记述法只有 cond 语句,但是因为只有这种记述语句时,对于正规的程序记述是不方便的,所以在实际的 Lisp 的处理系统中,通常提供了下列条件转移的记述方法:

- if 语句 与现有型语言的 if 语句的意义完全相同。
- when 语句和 unless 语句 现有型语言中无这种语句,它的意义是“条件成立时(when)或不成立时(unless),执行程序”。
- case 语句 它与 C 语言中的 switch~case 语句的意义相同。

2. 重 复

所谓重复,就是多次地执行同一个程序(如果有必要,可同时改变参数)。在 C 语言等现有型语言中,虽然存在着 for 语句、while 语句之类的记述方法,但在 Lisp 中却没有这种重复语句的记述方法。在 Lisp 中,是通过循环调用程序实现重复操作的。所谓循环调用,就是指“在定义某个程序时,在定义程序的过程中,调用其程序本身”。

例如,阶乘可以写成

$$n! = n \times (n-1) \times \cdots \times 1$$

可是,上式也可以写成

$$n! = \begin{cases} 1 & n=1 \text{ 时} \\ n \times (n-1)! & n \text{ 为 } 1 \text{ 以外的其他值时} \end{cases}$$

观察上式可以看出,在定义 $n!$ 时,最终还使用了 $(n-1)!$ 阶乘这个函数本身。如果把它转变成程序,则变成“在阶乘的定义中,调用阶乘这种程序”。在这种 Lisp 中,利用循环调用来实现通常的重复步骤。重复语句因为可以变换成循环调用,所以它们必定具有相同的效果。

下列程序是上述阶乘计算用 Lisp 表示的形式:

```
(defun factorial (num)
  (cond ((= num 1) 1)
        (t (* (factorial (- num 1)) num)))
)
```

在这个程序中,其意义这时变成为“如果自变量(num)的值为 1,即((cond ((= num 1))),则执行部分的 1 被设定为这个函数 factorial 的执行结果,如果不是这样,则应把从自变量(num)中减 1 的结果的阶乘值,与自变量相乘后的值((t (* (factorial (-

num 1)) num)))，设定为这个函数的执行结果”。

在这样的阶乘计算中，表示的重复过程，是一种作为 Lisp 数据的表而具有递归结构的程序。在具有递归结构的数据处理中，若应用循环调用，则程序可以自然地进行记述。因此，在 Lisp 中，是通过循环调用实现重复操作的。例如，当用图解的方法表示这个 factorial 的行为时，就变成了图 7.4 那种情况。“为了计算 (factorial 3)，调用 (factorial 2)，而为此又要调用 (factorial 1)”，这种情况应予以关注。

尽管如此，仍然与条件转移的情况相同，在实际的 Lisp 的处理系统中，提供了与现有型语言相似的有关重复的记述方法。例如在 Common Lisp 中，存在有 (loop<程序的表>) 这样的对重复的记述形式：

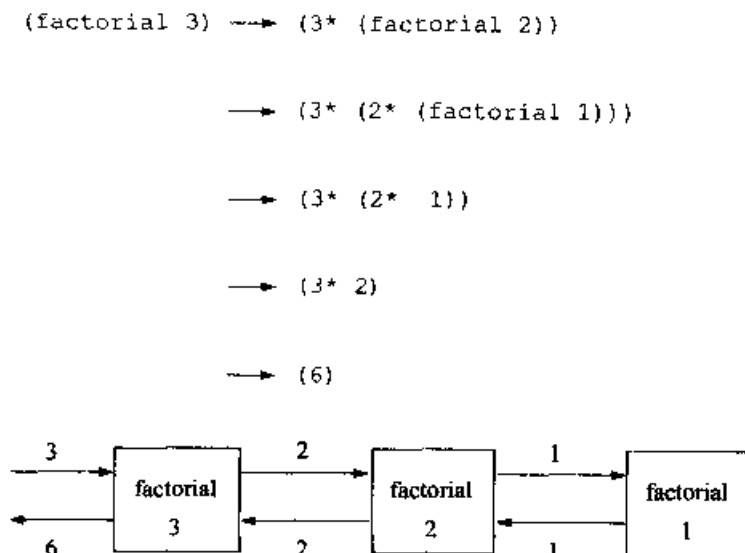


图 7.4 factorial 的函数循环调用情况

即使在其他情况下，诸如记述重复次数被指定的 dotimes，和能使用局部变量的 prog 系统的记述等，在实际的处理系统中，提供某个种类的重复的记述方法是平常的事。

7.2.5 其他的 Lisp 函数

在 Lisp 中，除了上述函数以外，还预先定义了其他许多函数。例如支配着输入和输出的 read 和 write 之类函数，进行数值运算的 +, -, *, / 等函数，还有用于逻辑运算的 and, or, not 等函数，凡此种种，均为已经存在的函数。然而当读者今后有机会用 Lisp 编写程序时，首先还是必须熟悉有关 Lisp 的手册。原因在于 Lisp

是一种有许多“方言”的语言。这里所说的“方言”，是指处理系统中语言的规定¹⁾有一些不同。Lisp 由于其语言特征，它可以简单地改变语言处理系统。因此，很多情况下，各位研究人员将根据自身的情况，朝着对自己有利的方向，改变 Lisp 的语言处理系统。针对这种情况，出现了“编制标准的 Lisp”的潮流。这种潮流的成果，就是称为 Common Lisp 的 Lisp。Common Lisp 汇集了当时许多 Lisp 处理系统的语言规范，可以说是集多种规范之大成，正是由于它具有这种性质，所以新的 Lisp 处理系统，多数都是以 Common Lisp 为依据²⁾作成的。

另一方面，Common Lisp 由于其性能，使得语言规范会变得很庞大，但是不可否认的是它还会失去一部分 Lisp 规范。因此，根据关于“更符号化 Lisp”的思考，制定了 Scheme 这种 Lisp（它是方言的一种）。根据对 Scheme 扩大化的 Common Lisp 的思考，语言规范是符号时，容易进行学习，另外，处理系统的工作也会变得比较容易³⁾。因此，在教育现场中，使用 Scheme 的地方也很多。

总之，因为在仍然是由处理系统提供的函数中，会有很多不同的情况，所以在实际编制程序时，我们应该仔细地阅读有关使用 Lisp 的处理系统的手册。

7.2.6 Lisp 的执行例子

最后，我们简单介绍在实际中 Lisp 程序的执行情况。为了能稍许感觉到符号处理的气氛，这里我们首先介绍用 Lisp 实施集合演算的情况。作为对象的演算有下列两项：

- 元素的确认 判定某个符号是不是集合的元素。
- 积集合的计算 两个集合的积集合的计算。

1. 基于 Lisp 的集合演算

首先从元素的确认程序开始。若把这个程序取名为 member，则 member 的程序变成为下列形式：

```
(defun member (element list_of_elements)      (1)
```

```
  (cond ((eq list_of_elements ()) nil)      (2)
```

```
        ((eq element (first list_of_element)) t)
```

```
      (3)
```

1) 若采用专门术语，则称其为语言规范。

2) 可以处理用 Common Lisp 记述的程序。

3) 例如 Common Lisp 的手册约为 1000 页，而相应的 Scheme 的手册则只有约 50 页。

```

(t (member element (rest list_of_elements)))
(4)

)

)

```

在上述程序的(1)中,宣告了这个程序的名称(member)和自变量(element list_of_elements)。其次,(2)表明意思是“给定的表如果是空表,则返回值为“假”(nil)”。这相应于“空集合不含任何元素”。再次,(3)表明意思是“自变量 list_of_elements 的第一个元素如果与 element 相等,则返回值为“真”(t)”。而(4)的意义则变为“如果不是这样,则应判定是否包含除自变量的第一个元素外,剩余的表中的元素”。这样就可以试着把它们实际地输入到处理系统中。

为了在处理系统中读入程序,通常采用 load 函数。目前,这种程序都被书写为 shuugou.lisp 之类文件,如果这样设定,则变成下列形式¹⁾。

```

$ (load 'shuugou.lisp)
; LOADING SHUUGOU.LISP
T
$

```

于是,处理系统便读入了程序。在这里,产生出了下列形式的输入:

```

$ (member 'a'(a b c))
T
$

```

于是,处理系统沿着 T 这个解答返回²⁾。因为 T 表示 true,也就是表示“真”,所以可以确认,a 这个原子就是(a b c)这个表的元素。其次,若设

```

$ (member 'b'(a b c))
T
$

```

则同样在解答 T 返回,显然,这也是正确的行动。再其次,若设

```

$ (member 'x'(a b c))

```

1) 实际上,表示成什么样的图像,因使用的处理系统不同而有所不同。

2) 通常在 Lisp 的处理系统中,大写字母和小写字母是没有区别的。因此系统的响应全都以大写字母输出,用户的输入也被变换成大写字母。

NIL

\$

则这回在解答 NIL 返回。这意味着“x 不是(a b c)的元素”。这样,判定第一自变量(表或原子)是不是第二自变量(表)的元素的任務,就由程序完成了。

然后,使用 member 函数,作计算交集(intersection)的程序。交集程序的基本考虑方法如下。

对于由第一自变量给出的表中的所有元素,重复地进行下列步骤。也就是说,如果元素被包含在第二自变量的表中,那么它必定包含有交集。如果不是这样,则利用除去该元素的表和原来的第二自变量的表,计算交集。

具体的交集的计算程序变成为下列形式:

```
(defun intersection (list1 list2)
  (cond ((atom list1) nil)
        ((member (first list1) list2)           (1)
         (cons (first list1) (intersection (rest list1)
                                             list2))) (2)
        (t (intersection (rest list1) list2)) (3)
         )
  )
)
```

执行上述程序,得到下列结果:

```
$ (intersection '(a b c) '(a d e))
(A)
$
```

这表示了“集合{a b c}与{a d e}的交集为{a}”。同样,

```
$ (intersection '(a b c) '(a e c))
(A C)
$
```

上述结果表示了“集合{a b c}与{a e c}的交集为{a c}”。另外

```
$ (intersection '(a b c) '(d e f))
NIL
$
```

上述结果表示了“集合{a b c}与{d e f}的交集{ }最终为空集”。在这个交集的程序中,包含了前面定义的 member 这个函数(上面程序表中的(1))。当交集的程序执行过程进展到这里时,前

面定义的 member 函数将被调出, T 或 NIL 的值进行返回。获得上述返回值的程序(2), 执行(3)中的哪一项被确定下来。这样在 Lisp 中, 就可以把小的函数组合起来, 构成完成更大功能的函数。

2. 基于 Lisp 的梵塔

下面再介绍一个例题。这个例题是在 AI 教科书中必定详细介绍的“梵塔”问题中的一个难题。因为求解这个难题是一种智力劳动, 所以可以有效地采用过去关于 AI 的研究成果。在这里我们希望以递归性为焦点, 对程序进行研究。

所谓梵塔问题, 是下列这样一种问题。

设在某处立有三根杆子, 并且有三个空心圆盘重叠着穿在杆子上。圆盘是自上而下地由小到大地重叠着置放的。现在欲将这些圆盘每次一个地移动, 从一根杆子转移到另一根杆子上。但是不能够把大圆盘重叠到小圆盘之上。

基本思路如下:

在把 n 个圆盘从最左边的杆上转移到最右边的杆上时, 先把 n 个圆盘上面的 $n-1$ 个圆盘从左边杆上转移到中间的杆上, 把左边杆上剩余的一个圆盘从左边转移到右边杆上。然后, 再将位于中间杆上的 $n-1$ 个圆盘转移至右边杆上。

这里应用了“在移动 n 个圆盘时, 首先转移 $n-1$ 个圆盘, 然后再转移剩余的一个圆盘”这种递归考虑方法。因此, 例如“在移动三个圆盘, 首先移动两个圆盘。再移动一个圆盘”这种工作程序便形成了, 由于移动圆盘的程序是共同的, 所以最终在移动圆盘的程序中, 会形成对自己本身的调用。图 7.5 表示了这种考虑方法。

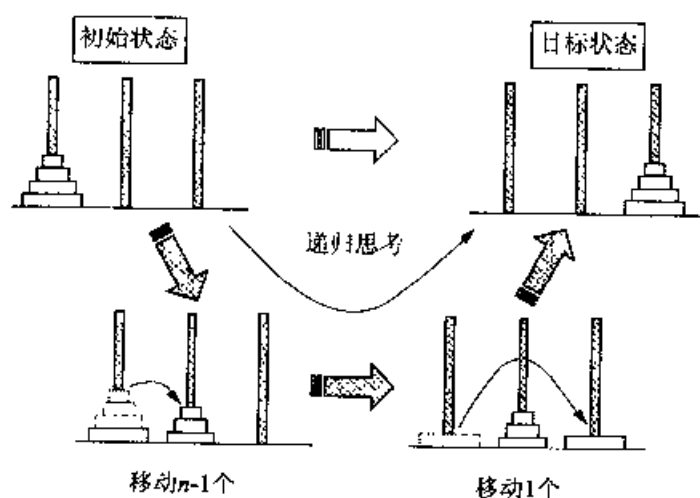


图 7.5 “梵塔”问题的考虑方法

程序变成下列形式¹⁾：

```
(defun hanoi (n) (transfer 'left 'right 'center n))
```

(1)

```
(defun transfer (from to work number)
```

(2)

```
(cond ((= number 1) (move-disk from to))
```

(3)

```
(t (append (transfer from work to (- number 1))
```

(4)

```
(move-disk from to)
```

(5)

```
(transfer work to from (- number 1)))
```

(6)

```
)
```

```
)
```

```
)
```

构成这个程序核心的函数是 transfer 这个函数。这个程序将实施下列一些运作：

(1) 是调出构成核心部分程序的函数。这里用 left, right, center 这三个原子表示三根杆。

(2) 中定义了 transfer 这个函数。这个函数采用了四个自变量, 最初的自变量表示移动前的杆, 第二号自变量表示移动后的杆, 第三号自变量表示工作用的杆, 第四号自变量表示圆盘的数量。于是程序的目标是从以 from 表示的杆把圆盘移动到以 to 表示的杆。

(3) 表示圆盘为一个时的处理。如果是一个圆盘, 则因为可以使该圆盘自由地移动, 所以它可以从 from 移动到 to。move disk 是移动圆盘的程序。虽然实际上利用印刷文字等的情况很多, 但是如果使用图示终端, 编写出能在画面上实际地移动圆盘那样的程序, 也许会更加精彩。

从(4)到(6)表示了圆盘数在两个以上时的处理。这时, 首先从上方将 $n-1$ 个圆盘从 from 移动到 work, 然后将剩余的一个圆盘从 from 移动到 to。最后进行将 work 的 $n-1$ 个圆盘移动到 to 的操作。因为这里将操作的结果用表的形式表示, 所以变成了下列形式。

1) 实际上启动这个程序时, 只有这些是不充分的。例如必须写出 move-disk 等有关程序, 但是构成核心的部分是 transfer 这个函数。

(4) 在这一步中,将 $n-1$ 个圆盘从 from 移动到 work。这里,在 transfer 这个现在定义的函数中,将实现调出自己本身的操作(循环调用),其次,transfer 的自变量将顺序地构成为 from, work, to 这种序列。因此,值得注意的是,最后表示圆盘个数的第四号中的数字,应比原来的个数再少一个。

(5) 因为在(4)的操作结果中,只剩下一个圆盘,所以将这一个圆盘从 from 移动到 to。

(6) 将在(4)的操作中移动至 work 的圆盘,变换转移到 to。值得注意的是,这里 transfer 的自变量变成为 work, to, from 这种序列。

从(4)到(6)是这个程序的核心。在这里巧妙地应用了递归考虑方法。编写这种递归程序,是因为在问题中给出的“重叠的圆盘”,具有递归的结构。

以上是用 Lisp 编写的梵塔的程序。由于利用了 Lisp 的表结构和递归考虑方法,所以可以把初看起来很复杂的难题,化为非常简单的程序,希望读者通过上述例题,能充分理解这个特点。

7.3 逻辑型语言 Prolog

本节将介绍逻辑型语言 Prolog。所谓逻辑型语言,是以形式逻辑或数理逻辑作为基础的程序设计语言。在逻辑型语言中,构成了“计算=证明”这样一种观点。这就是说,进行计算被看作是对逻辑式的证明。具体的说,它变成为下列内容。

现在若有 A 这个事实,例如若有“苏格拉底是人类”这个事实,和 $A \rightarrow B$ (读作“如果是 A 则是 B”)这个规则,例如“如果是人类则一定死”这个规则。那么这时利用三段推理, B 这个事实,也就是“苏格拉底一定死”这个事实,就可以被推导出来。这就是当把 A 这个事实和 $A \rightarrow B$ 这个规则作为公理系时,就相当于证明了 B 这个事实是正确的。这样一种利用三段推理法导出新的事实的做法,就相当于逻辑型语言的“计算”。

7.3.1 项:具有递归结构的另一种数据结构

相当于逻辑型语言的数据的内容,被称为项(term)。另外,相当于程序的内容则被称为逻辑式(formulae)。这两项内容相当于

描述逻辑型语言时的“措词”。首先从项的定义开始进行讨论。

所谓项是可以定义如下的一种符号列：

- 常量是项。
- 变量是项。
- f 是 n 个自变量的函数符号(函数的名称), t_1, \dots, t_n 分别为项时, $f(t_1, \dots, t_n)$ 是项。
- 仅由上述步骤定义的内容是项。

因此, 根据开始的两个定义, a , tanaka (田中), 1.5 等常量, 或者 x , ctemp ¹⁾ 等的变量, 都构成项。此外, 根据第三项定义, $\text{affiliate}(\text{tanaka}, \text{abc-company})$ 和 $\text{affiliate}(\text{ishida}(\text{石田}), x)$ 等构成的复杂形式的符号列, 以及 $\text{affiliate}(\text{name}(x, \text{tanaka}), \text{engineer})$ 这样的符号列, 它是既有作为项的自变量, 又有项的符号列, 它们也构成了项。因此, 项与 Lisp 的表是相似的。表可以包含作为其构成元素的表, 项也可以包含作为项的构成元素的项。总之, 项也和表一样, 具有递归的结构。

下面介绍逻辑式的定义, 首先介绍基本逻辑式的定义, 基本逻辑式是逻辑式定义的构成基础。

- 设 p 为谓词符号(谓词的名称), t_1, \dots, t_n 为项。这时称 $p(t_1, \dots, t_n)$ 为基本逻辑式(atomic formula)。

以这种基本逻辑式为基础, 对逻辑式定义如下:

- 基本逻辑式是逻辑式。
- 当存在有两个逻辑式 P 和 Q 时, 由这两个式子做成的下列式子也是逻辑式:
 $\neg P$ (P 的否定), $P \wedge Q$ (与), $P \vee Q$ (或), $P \Rightarrow Q$ (归结), $P \Leftrightarrow Q$ (同值), $\forall x P$ (对于所有 x , P 成立; 全称量词), $\exists x P$ (P 成立 x 存在; 存在量词)

- 只有上面构成的式子是逻辑式。

前面各项因为全都是基本逻辑式, 所以都是逻辑式。随后的两个式子是用 \wedge 和 \vee 等对逻辑式组合构成的式子, 所以也是逻辑式。另外, 若把 $P \wedge Q \wedge R$ 这种式子也考虑成 $(P \wedge Q) \wedge R$, 而 $(P \wedge Q)$ 是逻辑式, 所以这个逻辑式与同样是逻辑式的 R , 用算子 \wedge 结合起来得到的式子, 仍然是逻辑式。因此, 只要利用这个定义就能够定义任何复杂的逻辑式。

1) 正如在后面描述的那样, 在 Prolog 中, 变量用大写字母开头的一系列字符表示, 常量用一系列小写字符表示。

7.3.2 逻辑型语言的计算方法:归结原理

在逻辑语言中,把逻辑式的证明看作为计算。这时应怎样具体地进行证明呢?这种方法就是这里要介绍的归结原理(resolution principle)。归结原理就是由多个逻辑式,机械地构成新逻辑式的方法。由于发现了这种归结原理,使得逻辑型语言变成了实用的程序设计,从这一点看,这个归结原理是非常重要的。

归纳原理这种考虑方法本身,在某种意义上,是非常单纯朴素的方法。归纳原理的最朴素的方法是:

当存在有两个逻辑式 $A \rightarrow B$ 和 $B \rightarrow C$ 时

由这两个逻辑式可以导出 $A \rightarrow C$

这种方法从直觉上看是很明显的,如果涉及到一些道理,那么可以列举如下:

(1) $A \rightarrow B$ 与 $\neg A \vee B$ 是等价的。

(2) 由 $A \rightarrow B$ 和 $B \rightarrow C$,可以导出 $\neg A \vee B$ 和 $\neg B \vee C$ 。

(3) 当将 $\neg A \vee B$ 与 $\neg B \vee C$ 并列时,则 $\neg B$ 与 B 可消除并且合并导出 $\neg A \vee C$ 。

(4) $\neg A \vee C$ 与 $A \rightarrow C$ 相等。

根据上述理由,当存在逻辑式 $A \rightarrow B$ 和 $B \rightarrow C$ 时,由这两个式子显然可以导出 $A \rightarrow C$ 。而且利用这种步骤的重要意义,在于从(1)到(4)间的步骤,全都可以机械地进行。总之,它与人类的想法和智慧这样一些不确定因素完全无关,这些步骤可以纯粹机械地执行,这一点对于计算机来说是极为重要的。换句话说,“证明”这种初看起来属于人类智能的行为,实际上若从归纳原理这种观点去看,(虽然是在限定的范围内)就意味着能够纯粹机械地执行上述步骤。

如果用更简单一些的形式叙述上述归纳原理,则变成为下列形式:

- 将给定的公理系写成 $G = \{A, B, \dots, C\}$ 这种逻辑式集合的形式。

- 在这个公理系的基础上,要证明逻辑式 D 。这时,归纳原理采用了下列证明方法。设在给定的公理中,增加 D 的否定形 $\neg D$,构成新公理 $G' = \{A, B, \dots, C, \neg D\}$,考虑这个新公理,因为这个公理显示出包含着矛盾,所以 $\neg D$ 是错误的,因此 D 是正确的。

- 作为具体的证明步骤,证明过程经过下列一些步骤。首先从 G' 中提取出两个适当的逻辑式,构成一个适用于归纳原理的新

逻辑式¹⁾。重复进行这种操作,假定最后能够导出 P 和 $\neg P$ 。 P 与其否定形 $\neg P$ 双方存在这个事实,意味着哪一方都成立,这显然是矛盾的。总之,为了进行证明在原来的逻辑式中加进了逻辑式 D 的否定形,但却导致了矛盾,所以逻辑式 $\neg D$ 变成“假”,因此,逻辑式 D 变成为“真”²⁾。

在构成数理逻辑对象的逻辑中,有命题逻辑和谓词逻辑,它们有重大区别。所谓命题逻辑,是在其逻辑式中不包含变量的逻辑式。另一方面,所谓谓词逻辑,则是在其逻辑式中可以包含变量的逻辑体系。如果没有变量的概念,因为几乎不构成作为程序设计语言的意义,所以逻辑型语言把谓词逻辑作为基础理论应用。

前面的归纳原理,对于命题逻辑和谓词逻辑都成立。对于命题逻辑是成立的这个事实,由以前的说明可知是明显的,对于谓词逻辑,归纳原理也是成立的。但是,由于谓词逻辑的逻辑式中包含着变量,所以对朴素的归纳原理少许想些办法是必要的。对于前面的证明步骤,当由两个逻辑式构成一个新的逻辑式时,应选择哪种形式的逻辑式呢?为解决该问题应通过下列步骤,首先把适当的值代入包含在逻辑式中的变量中,构成 P 与 $\neg P$ 这样一对逻辑式,然后在此基础上,消去这两个逻辑式,归纳出一个新的逻辑式来。这种“把适当的值代入包含在逻辑式中的变量中”的操作是很重要的,根据这种操作,使得“求值”的要求得以实现。

【例】

设存在逻辑式 $\text{blonde}(a)$ 和 $\text{blond}(X) \rightarrow \text{blonde}(\text{father}(X))$ 。这时试考虑证明 $\text{blonde}(\text{father}(\text{father}(a)))$ 。例如,若设 $\text{blonde}(a)$ 意味着“ a 是金发人”这样的谓词,并设 father 为表示父亲关系的词,则 $\text{blond}(X) \rightarrow \text{blonde}(\text{father}(X))$ 与“如果有金发人,则其父亲也是金发人”这种事实是对应的³⁾。另外,证明对象的 $\text{blonde}(\text{father}(\text{father}(a)))$,则与“父亲的父亲即祖父也是金发人”是相当的。

上述证明步骤可以归纳如下:

(1) 首先整理公理系,将所有逻辑式变换成只包含 \wedge, \vee 与 \neg (否定)的形式。进而增加可以证明式子的否定形,构成公理系。在现在的情况下,分别变为下列形式:

- 1) 如果取前面的例子进行说明,则有由 $A \rightarrow B$ 和 $B \rightarrow C$ 可以导出 $A \rightarrow C$ 这个新逻辑式。
- 2) 总之,采用了反证法进行证明。
- 3) 这里的问题不涉及在遗传学上是否正确,纯粹是逻辑问题。

(i) blonde(a)

原来给出的公理系

(ii) $\neg \text{blonde}(X) \vee \text{blonde}(\text{father}(X))$

原来给出的公理系 $\text{blonde}(X) \rightarrow \text{blonde}(\text{father}(X))$ 被变形后的结果。

(iii) $\neg \text{blonde}(\text{father}(\text{father}(a)))$

可以证明的逻辑式 $\text{blonde}(\text{father}(\text{father}(a)))$ 的否定形。

(2) 证明步骤如下进行:

(a) 设 $X = a$, 由 (i) 和 (ii) 可以得到

$$\frac{\text{blonde}(a), \neg \text{blonde}(a) \vee \text{blonde}(\text{father}(a))}{\text{blonde}(\text{father}(a))}$$

(b) 根据 (a) 中的结果, 和在 (ii) 中设 $X = \text{father}(a)$ 后的结果, 可以得到

$$\frac{\text{blonde}(\text{father}(a)), \neg \text{blonde}(\text{father}(a)) \vee \text{blonde}(\text{father}(\text{father}(a)))}{\text{blonde}(\text{father}(\text{father}(a)))}$$

(c) 由 (b) 的结果和 (iii), 得到

$$\frac{\text{blonde}(\text{father}(\text{father}(a))), \neg \text{blonde}(\text{father}(\text{father}(a)))}{\square}$$

即导致了矛盾的产生。因为要证明的逻辑式的否定, 导致了包含有矛盾, 所以要证明的式子是成立的。

由这些步骤可以清楚地看出, 这种操作是可以纯粹地依照机械的方式进行的。这种基于归纳原理的证明步骤, 被用来作为程序设计语言的处理方式, 这就是逻辑型语言 Prolog。

7.3.3 Prolog 的对象: Horn 逻辑式

Prolog 虽然以谓词逻辑作为基础, 但是实际上并不是把一般的逻辑式作为对象, 而仅仅是把受到某种限制的逻辑式作为对象。具有这种 Prolog 对象的逻辑式, 称为 **Horn 逻辑式**。

一般的逻辑式具有下列形式:

$$A_1, \dots, A_m \rightarrow B_1, \dots, B_n$$

左边的 A_1, \dots, A_m 可以称为条件部分, 右边的 B_1, \dots, B_n 则可以称为结论部分, 但是对于 Horn 逻辑式, 右边的结论部分最多只能有一个逻辑式。这个限制初看起来是一种相当严厉的约束条件, 但是实际上并不是那么严重。

实际上, 如果从逻辑式的形式去描述问题, 多数问题的结论部分可以只用一个逻辑式描述。另外, 如果将 $A_1, \dots, A_m \rightarrow B_1, \dots,$

B_n 改写,则可以变成 $A_1 \wedge \cdots \wedge A_m \rightarrow B_1 \vee \cdots \vee B_n$ 这种形式,这意味着“结论部分的 B_1, \cdots, B_n 中,随便哪一个逻辑式都是成立的”,换句话说,这意味着“在 B_1, \cdots, B_n 中,没有明确说明哪一个逻辑式是成立的”,作为程序设计语言,它具有的这种性质是不能令人满意的。此外,由于把逻辑式的对象限定在 Horn 逻辑式,所以,众所周知存在着某种特殊的(高效的)处理手续¹⁾。因此,由于把对象限定于 Horn 逻辑式,从而使 Prolog 变成了一种具有程序设计语言“资格”的语言。

前面把 Horn 逻辑式定义为“右边的结论部分最多只能有一个逻辑式的情况”。那么这个所谓“最多”的意义是什么呢?实际上它是和 Prolog 程序的三种形式有关的。

(1) 首先考虑结论部分是一个(只有一个 B_1),条件部分有一个以上(A_1, \cdots, A_m)时的情况。这时的意义是“如果从 A_1 到 A_m 全都成立,则 B_1 成立。因为这意味着“如果(if)从 A_1 到 A_m 成立,则(then) B_1 成立”,所以这种形式的逻辑式被称为规则型逻辑式,或者采用“节”这个词,称之为规则节。形式上可以把规则节写成²⁾

$$B_1 \leftarrow A_1, \cdots, A_m$$

(2) 其次,考虑无条件部分的情况,即考虑 A_1, \cdots, A_m 均不存在,而结论部分只有一个逻辑式的情况。若从文字上解释这种情况,则意味着“在无任何条件的情况下,结论 B_1 成立”。换句话说,因为这意味着“ B_1 是无条件成立的”,所以考虑到表示的事实,这种形式的逻辑式,被称为事实型逻辑式,或者称之为事实节。事实节可以表示成下列形式:

$$B_1 \leftarrow$$

(3) 再次,考虑有条件部分,但无结论部分的情况,即考虑存在有 A_1, \cdots, A_m ,但不存在 B_1 部分时的情况。这意味着“从 A_1 到 A_m 如果成立,则无任何结论”。这种现象从直觉上看是很难理解的,但是作为逻辑性意义,它意味着“从 A_1 到 A_m 不同时成立”,或

1) 例如被称为 SPU(selective unit resolution),或者被称为 SNL(selective negative linear resolution)的处理手续。详细内容可参阅参考文献。

2) 以前表示逻辑式时,都是把条件部分写到左边,结论部分写到右边,箭头从左至右书写,但是在逻辑型语言中,都习惯于把结论部分写到左方,条件部分写到右方,箭头从右向左书写,以此来表示逻辑式。因此,从此以后,本书也将依照这种惯例进行表示。此前的表示方法只是方向不同而已,但包含的内容具有完全相同的意义。

者意味着“从 $\rightarrow A_1$ 到 $\rightarrow A_m$ 中的哪一个成立”¹⁾。从 Prolog 的执行情况看,习惯上称之为质问型逻辑式,或者称之为质问节。质问节可以写成下列形式:

$$\leftarrow A_1, \dots, A_m$$

(4) 前面介绍了 Prolog 的“三种类型”的形式。作为 Prolog 的程序形式虽然有上述三种类型,但是作为逻辑式的形态,还存在着另外一种。这就是“条件部分和结论部分全都不存在”这种形态。这可以表示成“ \leftarrow ”这种形式。这种形式的意义虽然在直观上不容易理解,但是在逻辑型语言中,这种形态被解释为表示矛盾的一种形式。

下面,在逻辑型语言领域,说明用这种形态表示矛盾的情况。

在逻辑型语言中,将欲证明的逻辑式的否定,附加到给定的公理系中,然后利用归纳原理导出矛盾,以此构成基本的计算机构。如果借助于上述从(1)到(4)中的措词来描述这个计算机构,那么就变成了下列内容:

“例如,在欲证明 $A_1 \wedge \dots \wedge A_n$ 的场合中,把这个逻辑式的否定,即把 $\leftarrow A_1, \dots, A_n$ 附加到节的集合中,则“ \leftarrow ”最终会导致矛盾发生”。

上述的内容,就是 Prolog 的基本计算机制。

7.3.4 Prolog 程序的表示法

Prolog 程序是依据 Prolog 程序设计语言文法,编写成的逻辑式的集合。Prolog 的程序一般按下列书写方式编写²⁾:

〈头部〉:—〈主体部〉

(1)〈头部〉为基本逻辑式。

(2)〈主体部〉与基本逻辑式并列。各基本逻辑式用逗号(,)隔开。

(3)符号:—表示 \leftarrow ,即表示“如果〈主体部〉成立,则〈头部〉也成立”

例如,下列表达式就是 Prolog 程序的例子:

$$a: \neg b, c, d.$$

这就是用 Prolog 的表示方法表示“ $a \leftarrow b, c, d$ ”这种逻辑式的

1) 这时利用 $B \leftarrow A$ 与 $\rightarrow A \vee B$ 是同值的这一事实,如果将 $\leftarrow A_1 \dots A_m$ 变换成 $\rightarrow A_1 \vee \dots \vee A_m$ 这种形式,问题就很清楚了。

2) 下示的书写方法是按照 DEC-10 Prolog,或者说是按照 Edinburgh Prolog 的书写方法编写的。

表示结果。因此,Prolog 的程序就变成了用这种表示方法表示的逻辑式的集合。

Prolog 的程序具有下列性质:

(1) 与〈主体部〉并列的基本逻辑式,全都可以用 \wedge (与) 结合起来。

(2) 具有相同〈头部〉的逻辑式,代表 \vee (或) 关系

因此,例如程序集合

$a: \neg b, c, d.$ (1)

$a: \neg e, f, g.$ (2)

$b.$ (3)

$c: \neg x, y.$ (4)

$c: \neg p, q.$ (5)

其意义如下:

用性质(1)表示的逻辑式的意义是“如果 b, c, d 同时成立,则 a 成立”。因为(1)与(2)具有相同的头部,所以构成为“或”的关系。若把(1)和(2)结合起来,则其意义变成“如果 b, c, d 或 e, f, g 成立,则 a 成立”。因为在(3)中没有主体部,所以也就变成了没有条件部,因此为无条件成立,即变成为事实。因为(4)和(5)也具有相同的头部,所以构成为“或”的关系。

7.3.5 单一化(unification)

在 Prolog 中,被称作单一化的功能起着重要作用。所谓单一化,就是“用适当的值代入两项中含有的变量内,使两项变成为一项的操作”。例如,当存在着 $p(a, X)$ 和 $p(Y, b)$ 这样两项时¹⁾,如果决定设 $X=b, Y=a$,那么两项就同时变为 $p(a, b)$ 这一项了。在这种包含有变量的情况中,以适当的值代入其变量后,使两项变成为一项的操作,就称为单一化。单一化有下列三种模式:

(1) 常量间的单一化 这只有在两个常量完全相同的情况下才能成功地进行单一化,其他情况下不能进行单一化。

(2) 常量和变量的单一化 这时可以进一步区分为两种情况:

- 在变量此前尚未与常量进行单一化的情况下,若以变量作

1) 这里也要遵循 Prolog 的程序设计惯例,假设用大写字母表示变量,用小写字母表示常量。则在这两项中, a 与 b 是常量, X 与 Y 是变量。

为单一化对象,并且将同一个常量代入到变量中,即可实现单一化。

- 在变量已经与常量进行了单一化的情况下¹⁾,则可依照(1)中相同的标准,判定单一化的成功与失败。

(3) 变量与变量的单一化 这时也可以进一步区分为两种情况:

- 在任一变量此前都未与常量实现单一化的情况下,由于可以设两个变量是相同的,从而成功地实现了单一化。

- 在一方或者两方的变量已经与别的值(常量)进行了单一化,从而使情况确定的条件下,则应与上述的(1),(2)相同,即应按照常量之间或者常量与变量之间单一化时的相同标准,判定单一化的失败与成功。

在进行单一化的对象的两项中,一部分内包含变量的情况下,可以将值(分别地)代入其每一个变量内,然后据此判定单一化的成败。但是,在一项中相同名称的变量应用于其他场所时(例如 $\text{affiliate}(X, \text{dept}(X))$),这时必须用相同的值代入变量中。

下面再少许介绍几个单一化的例子:

- 对于 $\text{blonde}(X)$ 和 $\text{blonde}(\text{father}(\text{jack}))$,当设 $X = \text{father}(\text{jack})$ 时,单一化是可能的。

- 对于 $\text{blonde}(X)$ 和 $\text{blonde}(\text{father}(\text{who}))$,当设 $X = \text{father}(\text{who})$ 时,单一化是可能的。这时,who 是变量,这样进行单一化,即使双方都包含变量,也不会出现问题。

- 关于 $p(X, X)$ 和 $p(1, Y)$ 的单一化,可根据下列方式进行。首先比较双方的第一自变量,因为它们分别为 X 和 1 ,所以当设 $X = 1$ 时,单一化即获得成功。其次当比较第二自变量时,因为它们分别是 X 和 Y ,所以当设 $X = Y$ 时,单一化即获得成功,但是因为在第一自变量单一化时,已经将 1 这个值代进 X 中,所以由于假设了 $X = Y$, 1 这个值也被代进了 Y 这个变量中。

这种 Prolog 的单一化机能,具有非常重要的作用。在 Prolog 中,通过单一化可以实现下列功能。

1. 情况区分

在常量之间单一化时,若双方不相同,则不能成功地进行单一化,利用这个性质,在 Prolog 中,通过单一化可以实现情况的区

1) 称这种情况为“变量被常量所制约”。

分。例如考虑程序“自变量为 a 时, 执行 p, q 这种谓词, 自变量为 b 时, 执行 r, s 这种谓词”。例如若用现有型语言, 它即使写成下列形式:

```
if (arg = 'a') {call(p); call(q);}
else {call(r); call(s);}
```

但是在 Prolog 中, 仍可以用下列形式描述:

```
pred(a): - p, q.
pred(b): - r, s.
```

例如对于这个谓词, 当假设执行 $\text{pred}(a)$ 这个调用时, 单一化结果调出的谓词和最初的节 $\text{pred}(a): - p, q.$ 的头部的单一化取得成功。其结果是主体部的两个谓词 p, q 被执行。 $\text{pred}(b)$ 的情况也是相同的。另一方面, 例如当假设执行 $\text{pred}(c)$ 这种调用时, 因为哪一节的单一化都不会成功, 所以 pred 这个谓词不被调用。这样, 在 prolog 中, 利用单一化就实现了情况的区分。

2. 自变量的交接和结果的返回

在 Prolog 中, 利用变量与常量间的单一化, 可以实现进行计算时参数的交接, 同时通过程序的调用, 可以使计算结果返回至初始位置。例如考虑下列程序:

```
pred(X, Y): - p(X, Z), q(Z, Y).
```

当以 $\text{pred}(a, p)$ 这种形式调用这个程序时, 定义一方的谓词的第一自变量 X 和调用一方的第一自变量 a 被单一化。因为变量与常量的单一化是常量代入到变量中, 所以定义一方的 X 被 a 这个值取代。最终, 计算用的参量被从调用一方交接到定义一方。

另一方面, 由于第二自变量是共同的变量, 所以它是变量间的单一化, 但是因为两个变量可以看作是相同的, 因此单一化一定成功。当这个程序被设计成这样一种功能的程序时, 即它能够“利用作为自变量被交付的 X , 令 p 这个谓词进行某种计算, 并把计算结果代入到 p 的第二个自变量中。然后把上述第二个自变量的值作为输入量, 使 q 这个谓词进行某种计算, 其计算结果交付给 q 的第二自变量”, 这样一来, 就成了将 $\text{pred}(X, Y)$ 的最终结果代入到了第二自变量。可是, 在最初的单一化时, 因为调用一方的变量 P 与定义一方的变量 Y 被单一化, 因而变成为同一变量, 所以代入到变量 Y 中的值, 与代入到 P 中的值是相同的。因此, 计算的结果通过单一化操作而被返还到调用一方。

这种参数间参数值的移动情况, 如图 7.6 所示。在图 7.6 中,

参数值沿着从①到⑤的顺序依次进行传送,最终,通过⑤的运行,将执行结果的值代入到调用一方的元素 P 这个变量中。

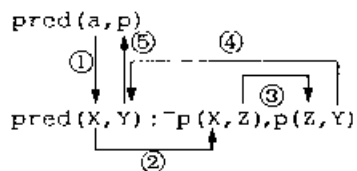


图 7.6 单一化的数据流动

通过上述例子可以看出,在 Prolog 中许多工作是通过“单一化”操作完成的。

7.3.6 Prolog 的表处理

在 Lisp 中,为了方便地处理“表”这种数据结构,准备了各种各样的机构。在利用 Prolog 构造人工智能系统时,“表”也具有重要作用。因此,在利用 Prolog 的“表”处理时,也会变得比较方便。

首先,作为“表”的表示方法,Prolog 的情况与 Lisp 非常相似。在 Prolog 中,可以通过下列方式表示“表”:

- 无任何元素的表(空表)在 Prolog 中用 `[]` 表示。
- 元素为 a 的表表示为 `[a]`。
- 元素为 a, b 和 c 的表,表示为 `[a, b, c]`。

最后, Lisp 和 Prolog 的“表”的表示方法,有下列不同:

- 在 Lisp 中,表的全体用“(”和“)”包围起来,而在 Prolog 中,则用“[”和“]”包围起来。
- 在 Lisp 中,利用空白分隔开“表”的元素,而在 Prolog 中则利用逗号“,”分隔开“表”的元素。

Lisp 和 Prolog 表的表示方法上,大的区别就是上述两点,此外,在表示上没有大的区别。另外,“表”还具有递归结构性质,也就是说,“表”可以包含作为“表”的元素的“表”,例如 `[a, [b, c], d]` 这种表示是可以的。

那么, Lisp 的 `first` 和 `rest`,或者 `cons` 等这样一些基本操作,在 Prolog 中又该怎样处理呢? 实际上这些操作全都可以通过单一化来实现。首先让我们来学习“表”的下列表示方法。

表的初始元素即在 Lisp 中称之为 `first` 的元素是 X , 剩余的元素即在 Lisp 中称之为 `rest` 的元素是 Y , 这两种元素构成的表可以

表示为 $[X|Y]$ 。

例如初始(first)元素为 a , 剩余的元素(rest)为 $[b,c]$, 这时的表可以表示为 $[a|[b,c]]$ 。若将这种表示进行复原, 即从表中取出初始元素 first, 和初始元素以外的元素 rest, 或者构成以谓词描述的 cons, 则会变成下列形式:

```
first first([X|Y], X).
```

```
rest rest([X|Y], Y).
```

```
cons cons(X, Y, [X|Y]).
```

例如, first 这种谓词的第一自变量, 是取出初始元素的输入自变量, 第二自变量是将取出的元素代入的输出自变量。这个谓词依照下列方式使用:

```
first([a,b,c], P).
```

若执行这个谓词, 根据单一化, a 将被代入定义一方的 first $([X|Y], X)$ 中的 X 。这作为程序, 将会使作为 first $([X|Y], X)$ 的第一自变量的 $[X|Y]$, 与调用一方的第一自变量 $[a,b,c]$ 进行单一化, 其结果是将 $[a,b,c]$ 中作为 first 的 a 单一化为 X , 将作为 rest 的 $[b,c]$ 单一化为 Y ¹⁾。因此, a 被代入到调用一方作为输出变量的 p 中。rest 和 cons 也会产生同样的运行过程。

在实际的程序中, 不特意地使用 first 和 rest 这样的谓词, 通常都是采用单一化处理。例如, 取出作为自变量给出的表的第一自变量, 并且将该第一自变量交付给别的谓词, 那么这时应作什么样的处理呢? 这时的谓词变为下列程序:

```
pred1 ([X|Y]): - pred2(X).
```

在这个谓词 pred1 的头部, 输入自变量(表)通过单一化而被分解成 first 的元素 X 和 rest 的元素 Y 。而且在主体部中, 因头部的单一化而被分解出 first 的元素, 将被传送给处理谓词 pred2 进行处理。

这样一来, 在 Prolog 中, 不用有意地采用特殊的函数(谓词), 进行取出“表”的先头部分或者取出“表”的先头部分以外元素的操作, 这些都可以通过单一化来处理。因此, Prolog 比 Lisp 能更为简洁地对表处理进行描述。

7.3.7 Prolog 的执行控制功能: 自动回溯和截断符号

这里我们来介绍 Prolog 的执行控制功能。

1) $[a,b,c]=[a|[b|c]]$ 。

1. 自动回溯

“自动回溯(auto backtracking)”是 Prolog 程序执行机构的支柱之一。例如,考虑下列程序:

$$p: -q, r, s.$$

$$q: -x, y, z.$$

$$q: -l, m, n.$$

这个程序显示出:

- 如果 q, r, s 成立, 则 p 成立(与关系)。
- x, y, z 成立, 如果 l, m, n 成立, 则 q 成立(或关系)。当 p 这个谓词被调用时, Prolog 的处理系统将依照下列进程执行程序。

(1) 最初被调用使用的谓词的单一化, 是通过对可能具有头部的逻辑式, 从上到下地进行搜索实现的。如果是在上述例子中, 则可以看出是 $p: -q, r, s.$ 。

(2) 在看到的逻辑式的主体部分中, 对于排列着的基本逻辑式, 应从左向右顺序地予以执行。如果针对当前的情况, 则依照 q, r, s 的顺序予以执行。

(3) 在执行 q 的情况中, 也要对 q 的具有可能进行单一化的头部的逻辑式进行搜索。在目前的情况下, $q: -x, y, z$ 和 $q: -l, m, n.$ 中的两个逻辑式的头部是可能单一化的。这时, Prolog 的处理系统最初执行前面见到的方案, 而最终对于这个例子, 则应执行 $q: -x, y, z.$ 。

对上述执行过程进行整理, 就变为下列内容。

- 对谓词的主体部分, 从左向右地予以执行。执行的结果为“真”即执行成功; 执行的结果为“假”即执行失败, 而且执行会返回到其中的某种结果。然而只有在前面的谓词执行成功时, 才转移去执行下一个谓词。

• 在可能对复数的逻辑式的头部单一化的情况中, 仍可按照前面找到的方法进行, 即从上面列写的逻辑式到从下面列写的逻辑式, 依次地予以执行(自上而下地执行)。

那么, 比如在执行 $p: -q, r, s.$ 的过程中, 在执行 r 时遭到失败, 最终得到逻辑上的“假”这种答案, 在这种情况下, Prolog 的处理系统应作些什么变动呢? 这时的 Prolog 处理系统, 应当在执行失败的谓词前面, 再一次地重新设置谓词。这里的“重新设置”, 其意思不是通常的“再计算”的意思, 而是“在或的关系中, 选择某一个别的逻辑式”这样一种意思。就上例而言, 例如在执行 r 的过程

中遭到失败,那么这时就要重新设置一个前面的 q 谓词,重新设置的意思是“在或的关系中选择某个别的分枝”这一层意思,因此,因为最初选择了 $q: -x, y, z$. 所以在选择了 $q: -1, m, n$. 时,就对这个谓词进行了再一次地重新设置。

若把这个过程作为程序的源码变动来观察,它就变成了下列过程。先执行 q , 然后执行右面的某个 r , 当其结果失败时,返回到原地再一次重新设置 q 。这种“返回原地”部分被称为自动回溯,或称为回溯。因此,在 Prolog 的处理系统中,用户即使不进行程序设计,也能够自动地重新进行失败后的计算。

2. 截断符号

自动回溯的功能虽然是一种非常有用的功能,但是有时人们意识到会产生不允许回溯的情况。用来控制这种回溯的谓词就是**截断符号**。截断符号在使用上与普通的谓词是相同的。它只是使用了一种特殊的符号“!”,其他与普通的谓词是相同的。例如,当在前面的程序中加进截断符号时,它就变成了下列形式:

$p: -q, !, r, s.$

$q: -x, y, z.$

$q: -1, m, n.$

这里,在 q 与 r 之间,插入了截断符号。截断符号是一定成功谓词,其逻辑意义与 $true$ 这个谓词完全相同。可是在执行顺序上意义是不同的。截断符号的作用是“发生回溯时,使回溯能返回到截断符号的位置,而不能重新访问截断符号以后(左面)的谓词”。因此,通过在适当的位置插入截断符号,可以完成对回溯作用的控制。

至于这个谓词(符号)为什么会被称为截断符号,是因为当执行这个谓词时,它具有下列功能,即“在这个谓词执行前,当被选择的 V (或) 的选择分枝全被确定下来,并且进行回溯时,不再选择其他“或”的选择分枝,换句话说,“或”的其他选择分枝被切除(剪切)”。截断符号在调出包含截断符号的节的谓词(它被称为母目标)执行后,确定所有的选择分枝。

由于这种截断符号的导入,使得诸如 $not(P)$ 这种谓词可以被定义。 $not(P)$ 这种谓词,是这样一种谓词,它意味着“当 P 成功时,它失败,而当 P 失败时,它成功”。其程序变成为下列形式¹⁾:

$not(P): -call(P), !, fail. \quad (1)$

1) 在实际的 Prolog 处理系统中,多数情况下 not 是作为内部谓词提供的。

not(P). (2)

在(1)的主体部的 $\text{call}(P)^{1)}$ 之后插入截断符号。这时这个程序的工作情况如下。

如果 $\text{call}(P)$ 成功,则其后的截断符号便被执行。因为这是一定会成功的,所以其后的 fail 这个谓词会被执行,而这又是一个一定失败的谓词,所以在这里会发生回溯。可是因为存在着截断符号,所以在截断符号处回溯会被终止,并且不能再返回访问截断符号前面的原来的子句。因此,作为程序的全体变成为失败,因此,实现了“如果 P 成功则失败”。

如果 $\text{call}(P)$ 失败,另外一个选择分枝(2)将被执行。因为这里不存在主体部(条件),所以一定会成功。因为执行(2)时,是在 $\text{call}(P)$ 失败的时候,所以这时就成为实现“如果 P 失败则成功”这样一种结果。

这种截断符号,作为程序语言,为了具备实用功能,就成为了一种必不可少的内容。但是从逻辑观点看,截断符号却变成了一种“不正常”的因素。由于存在这种截断符号,所以丧失了逻辑上的完备性。例如,在截断符号阻止回溯的地方,会发生下列情况,这就是由于执行截断符号,虽然在被剪切掉的分枝中包含着存在解答的可能性,但是由于有截断符号的作用,从而剪切掉了有可能包含着解答的选择分枝,所以尽管在逻辑上是正确的程序,但是却不能够得出解答。因此,必须注意,不能任意地滥用截断符号,如能巧妙地利用截断符号,就有可能构成具有良好效率的程序。

7.3.8 把程序变成数据,把数据变成程序: assert, retract 及 univ

在 Lisp 中,已经介绍过,由于采用了 eval 和 quote,可以使程序和数据自由地进行变换。同样的变换也可以在 Prolog 中进行。Prolog 的程序是逻辑式(节)的集合,在程序的执行过程中,既可以自由地增加这些节,也可以自由地消除这些节。另外,伴随着程序的运行,可以在程序中形成新的逻辑式。为此采用的谓词是 assert, retract 以及 univ 这三种谓词。

- assert 谓词 只要选取作成逻辑式形式的自变量,那么 assert 就是将该逻辑式,即 Prolog 的程序,进行追加的谓词。例如

1) call 这个谓词,是“把变成自变量的 P 设想为谓词来执行”的一种谓词,它与 Lisp 中的 eval 相对应。

若设¹⁾

```
assert(':-'(a,b,c)).
```

那么即可将 $a:-b,c$ 这种程序(逻辑式)追加到正在执行中的程序(逻辑式的集合)中。

• retract 谓词 只要选取作成逻辑式形式的自变量,那么 retract 就是将该逻辑式,即 Prolog 的程序从现在执行的程序中消除掉的谓词。例如若设

```
retract(':-'(a,b,c)).
```

则将会从现在执行中的程序(逻辑式的集合)中,把 $a:-b,c$ 这种程序(逻辑式)消除掉。

如果利用上述两种谓词,就会使自由且动态地改变现在执行中的程序成为可能。

此外,伴随着程序的运行,为了能动态地构成追加程序所采用的谓词,是一种被称为 univ 的谓词。习惯上采用“=..”这种记述形式,univ 一般采用下列形式:

```
C = ..[X|Y]
```

它的意义如下:

C 表示变量,该变量表示用 univ 形成的谓词,(表的 first 部分的)X 表示形成的谓词的名称,(表的 rest 部分的)Y 表示谓词的自变量。

因此,例如若要执行程序:

```
c = ..[pred,a,b].
```

则在 C 这个变量中,就被代进了 $\text{pred}(a,b)$ 这种(事实型的)逻辑式。另外,在生成规则型的逻辑式时,若设

```
C = ..[':-',pred(X,Y),a(X,Z),b(Z,Y)].
```

则在 C 中被代进了 $\text{pred}(X,Y):-a(Z,Y),b(Z,Y)$ 这种规则型式的逻辑式。

这样构成的逻辑式,因为通过利用前面的 assert 谓词,可以追加到表示程序的逻辑式集合中,因此,可以动态地构成程序,即动态地构成逻辑式,从而可以改变执行中的程序的运行情况。

1) ':-'(a,b,c) 这种记述形式,虽然构成了有些不寻常的形式,但是如果考虑到下列情况,它显然变成了一种相当优良的逻辑式。这就是':-'可以看作是谓词名称,a,b,c 可以看作是自变量。这种谓词实际上表示了 $a:-b,c$ 这种规则型的逻辑式。规则型的逻辑式,实际上是具有':-'这种标称的谓词名称的事实型逻辑式。总之, $a:-b,c,d$ 这种逻辑式,可以表示为':-'(a,b,c,d)。这种事实型的逻辑式。

7.3.9 其他的谓词

实际上,为了构成 Prolog 程序,除了以前介绍的各种谓词以外,还需要其他一些谓词。在实际的 Prolog 的处理系统中,这些谓词是以内部谓词(build-in predicates, BIPS)的形式提供的。这种 BIPS 虽然实际上因处理系统不同而存在微妙的差异,但是大致上下列谓词在大部分处理系统中都是会被提供的。

- 输入输出用的谓词 为了读写项的数据的谓词(read, write)和输入输出文字数据的谓词(get, put)等,将被提供。

- 算术运算用的谓词 Prolog 的运算构成了少许特殊的形式。在 Prolog 中,算术运算一般变成为<变量/常量>is<算术运算式>。例如若只写 $3+4$ 时,则这时表达的模式是“ $3+4$ ”,也就是“3 的后面接+这个符号,然后再接 4 这个符号”,不进行运算。因此,当要进行实际的运算时,必须有 is 这个谓词。通常在 Prolog 的处理系统中,进行算术运算+, -, *, /(加减乘除)时,为了能实际地进行算术运算, is 这种谓词等将会被提供应用。

- 执行控制 为了能够执行控制,在执行用前面介绍的截断符号! 和 univ 构成的谓词时,提供了 call 等谓词。

除此以外,还有一些情况是提供了诸如进行文字处理的谓词,以及利用 OS 功能的谓词等,因为在各种处理系统中存在着许多不同的情况,所以和在 Lisp 时的情况一样,在实际应用时,精读处理系统的手册是很重要的。

7.3.10 Prolog 的执行例子

在这里我们简单地介绍一下,实际中 Prolog 程序的执行情况。这里与 Lisp 情况时一样,进行同样的集合运算,用 Prolog 对梵塔进行处理。

1. 基于 Prolog 的集合运算

作为对象的运算,具有下列两类:

- 元素的确认 判定某个符号是不是集合的元素。
- 交集的计算 计算两个集合的交集

首先,从元素的确认程序开始进行说明。若把这个程序取名为 member,则 member 的程序变成为下列形式:

$$\text{member}(X, [X|Y]). \quad (1)$$

$$\text{member}(X, [X|Z]: \neg \text{member}(X, Z). \quad (2)$$

谓词 member 具有两个自变量。最初的自变量是比较对象的

元素,其后的第二号自变量则是表。然后,确定在第二号的表中,是否包含在第一号自变量中给出的元素。谓词 `member` 由两节构成。在最初的节中,第二号自变量(这里放上处理对象的表)的最初元素($[X|Y]$)中的 X ,和第一号自变量(这里放上确认存在项的数据)如果相等,那么就什么也不必作了。也就是说取得了成功。(2)表示了“不是(1)的情况”。在这种情况下,取作为第二自变量给出的表的 `rest`,构成确认以第一自变量给出的元素是否被包含在内的程序。

在处理系统中读取程序时,通常采用 `consult` 这个谓词。现在如果把把这个程序书写成 `shuugou.Prolog` 这种文件,那么就会变成下列形式¹⁾:

```
? - consult('shuugou.prolog').
      shuugou.pl   consulted
yes
? -
```

这相当于提出这样的质问,即“原子 b 包含在表 $[a,b,c]$ 中了吗”。于是,处理系统返回到 `yes` 这个回答上。因为 `yes` 表示真,所以可以确认 b 这个原子是 $[a,b,c]$ 这个表的元素。这时处理系统将依照下列方式运行。

- 对质问节 `member(b,[a,b,c])`,与最初书写的某个程序(1)的头部(`member(X,[X|Y])`)进行单一化。

- 因为第一自变量是常量(b)和变量 X 的单一化,所以通过设 $X=b$ 一定成功。

- 第二自变量是 $[a,b,c]$ 和 $[X|Y]$ 。在这种情况下,利用表之间的单一化,调用一方的 `first` 即 a 与程序一方的 `first` 即 X ,以及调用一方的 `rest` 即 $[b,c]$ 与程序一方作为 `rest` 的 Y ,均被进行了单一化。但是,这时因为与在第一步中被单一化时的 $X=b$ 相矛盾,所以单一化失败。

- 于是处理系统便采取另一种可能性,即执行 `member` 的或选择分枝((2)中的程序)。就是这时,也要首先尝试对头部的单一化。在这种情况下,通过在 X 中用 b , Y 中用 a , z 中用 $[b,c]$ 的方法,实施了单一化进程,因此头部的单一化获得成功。于是主体部便以 `member(b,[b,c])` 这种形式被调用。若以与前面的例子相同

1) 因为这也是与 Lisp 时的情况相同的处理系统,但输出形式不同。

的方法考虑该例,即可获得成功。因此,由于主体部是成功的,所以作为程序的全体也变成为成功的。

其次,如果设

? - member(x,[a,b,c]).

no

? -

那么这回将返回到 no 这个解答。这意味着“X 不是[a,b,c]的元素”。

在 Prolog 中,采用对这个元素进行确认的程序,这时只要取出集合中的元素就可以了。例如对于下列输入:

? - member(x,[a,b,c]).

在上式中,因为 X 为大写字母,所以表示变量。于是处理系统返回到下列解答:

X = a

yes

? -

最后当然会取出[a,b,c]这个集合中的元素,并且代入到变量中。总之,存在着这样一种有趣的性质,这就是“在 Prolog 的变量中,输入与输出没有区别”。这是由作为 Prolog 计算机构的单一化性质造成的。在单一化的时候,曾经谈到“自变量的交接和计算结果的返回”,正是这个特性导致了输入输出没有区别这个性质。

其次,应用 member 这个谓词,构造计算交集(intersection)的程序。交集程序的基本考虑方法如下。

对于在第一自变量中给出的表中的所有元素,进行下列重复操作。最后,如果元素被包含在第二自变量的表中,那么元素就应该包含在交集中。如果不是这样,则利用除去这些元素的表,与原来的第二自变量的表,计算交集。

具体的交集的程序变成为下列形式:

intersection([],X,[]). (1)

intersection([X|Y],Set,[X|Int]):- (2)

member(X,Set),!,intersection(Y,Set,Int). (3)

intersection([X|Y],Set,Int):- (4)

intersection(Y,Set,Int). (5)

在上述程序中,(1)表示“空集合与任意集合的交集为空集合”。(2)的意义是,“如果初始表中的最初元素([X|Y]的 X)是第二号表中的元素(member(X,set)),取初始表中剩余元素(Y)与第二号表的交集(intersection(Y,Set,Int)),并且把最初元素加到上述结果(Int)中([X|Int])”。在(2)的节的头部,若利用单一化取出表中的元素,则可以进行合成([X|Int]),这点希望能予以关注。最终,把作为(3)的最后的谓词 intersection(Y,Set,Int)的解答 Int,与作为程序调用一方的表[X|Y]的最初元素 X 组合起来,构成新的表[X|Int],并用这里的谓词调出上述结果,作为全体的解答。(4)是在(2)或者(3)的谓词失败时,也就是在第一自变量的表中的先头元素 X,不是第二自变量的表中的元素时,被调用的。这时,对除去第一自变量先头元素的剩余表 Y,与第二自变量的表,进行交集合计算,其解答 Int 就构成为全体的解答。

在这个 intersection 的程序中,包含着前面定义过的 member 这个谓词(上面的程序中的(3))。若进行到现在来执行 intersection 的程序,则前面定义的 member 谓词会被调用,并且返回到 yes 或者 no 的值。即使对于这样的 Prolog,也和 Lisp 一样,可以把小的谓词进行组合,从而构成完成更大功能的谓词。

2. 基于 Prolog 的梵塔

下面我们用 Prolog 再一次对求解梵塔的程序进行描述,其描述形式如下:

hanoi(N):-transfer(N,a,b,c). (1)

transfer(0,_,_,_). (2)

transfer(N,From,To,Work):-
N1 is N-1,transfer(N1,From,Work,To),), (3)

print(N,From,To), (4)

transfer(N1,Work,To,From). (5)

print(N,A,B):- (6)

write('Move'),write(N),write('th plate from'),
write(A),write('to'),write(B),nl.

Prolog 的程序也和 Lisp 情况时一样,采用循环调用方式。它体现在(3)的部分中。这里也实施了与 Lisp 情况相同的考虑方法。这种考虑方法是:

当把 n 个圆盘从最初左方的杆子向右方的杆子转换时,先将

n 个圆盘中上方的 $n-1$ 圆盘,从左方的杆子转换到中间的杆子上,然后再将左方杆子上剩下的一个圆盘,从左方杆子转换到右方杆子上。其后,再把位于中央杆子上的 $n-1$ 个圆盘,从中间的杆子转换到右方杆子上。

因此,在(3)中首先从作为第一自变量给出的圆盘个数(N)中减去 1,并将其作为自变量调用 transfer 这个谓词。这里表示杆子位置的 From, To, Work 这三个自变量的顺序,是从 From, To, Work 向着 From, Work, To 轮流地进行变化的,这一点希望读者予以注意。(4)表示 $n-1$ 个圆盘转移后,对剩下的一个圆盘的移动。因为在(3)中从 From 向 Work 移动 $n-1$ 个圆盘,所以其后的(5)表示将 From 作为工作范围,从 Work 向 To 移动 $n-1$ 圆盘的移动。此外,(6)是用来表示结果的谓词。

这里用文字形式进行了表示,但是和 Lisp 时的情况一样,用图解的形式表示程序,也是一件很有意思的事情。总之,即使对于 Prolog,也和 Lisp 的情况一样,数据与程序双方的递归构造,本质上起到了重要作用,这是显而易见的。

练习题

1. 试列举你认为作为人工智能系统的程序设计语言,必须具有的功能,并且加以说明。
2. 试述递归(recursive)构造的特点,并且说明它在人工智能系统方面起到的作用。
3. Lisp 中的 eval 和 quote 与 Prolog 中的 assert, retract, univ, 分别具有相同的功能。试述它们的功能,并且说明它们在人工智能系统方面起到的作用。
4. 对于 Lisp 的表处理,提供了提取最初元素的函数 first, 和提取最初以外的元素的函数 rest。试利用这两种函数的组合,构成提取第二号元素的函数 second 的程序。此外,试构成提取任意的第 n 号元素的函数 n -th 的程序。
5. Prolog 的表处理与 Lisp 不同,它全部都进行单一化(unification)处理。试构成提取第二号元素的谓词 second 和任意的第 n 号元素的谓词 n -th 的程序。

练习题简答

第 2 章

1.

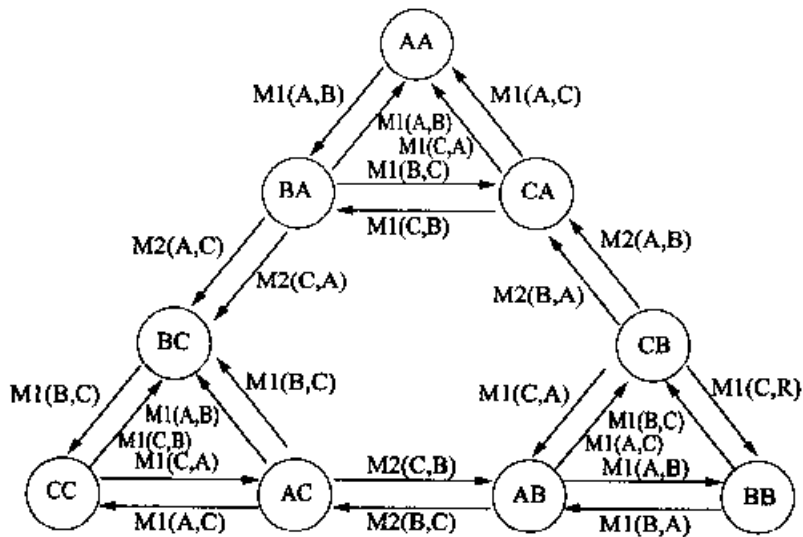


图 1

2. 下面表示了被展开节点的顺序。

纵向优先: $A \rightarrow B \rightarrow C \rightarrow D \rightarrow E$

横向优先: $A \rightarrow B \rightarrow H \rightarrow C \rightarrow G \rightarrow D \rightarrow F \rightarrow E$

均一代价: $A \rightarrow B \rightarrow H \rightarrow G \rightarrow C \rightarrow F \rightarrow D \rightarrow E$

最佳优先: $A \rightarrow H \rightarrow G \rightarrow D \rightarrow E$

A* 算法: $A \rightarrow H \rightarrow G \rightarrow F \rightarrow D \rightarrow E$

3. 下面表示了被展开节点的顺序。

$A \rightarrow D \rightarrow C \rightarrow D \rightarrow B \rightarrow D \rightarrow C \rightarrow D \rightarrow E$

4. 下面表示了 AND/OR 图和解图(粗线)。

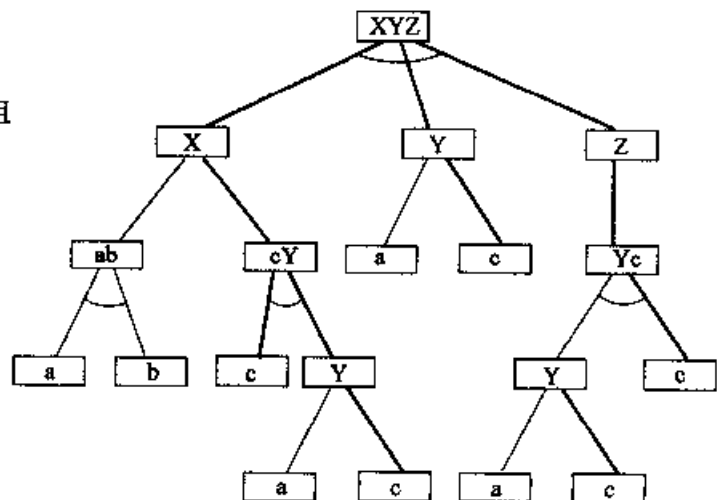


图 2

5.

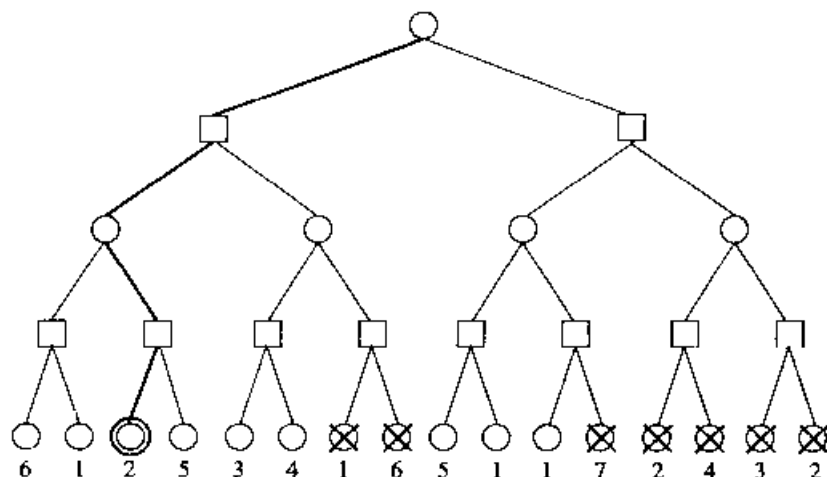


图 3

6. 解答略。

第 3 章

1. 产生式系统具有以下特征：

- (1) 便于处理未经整理的庞大的知识。
- (2) 能够简单地对应知识的追加,修正和清除。

2. (1) 变成为下列动作：

循环	冲突集合	使用规则	得到的数据
1	{(R2, {D1})}	R1	D5(太郎是哺乳动物)
2	{(R2, {D1}), (R5, {D2, D5})}	R5	D6(太郎是食肉动物)
3	{(R2, {D1}), (R5, {D2, D5}), (R8, {D3, D4, D6})}	R8	D7(太郎是老虎)

(2) 得到了图 4 所示的推理网络。

3. (1) 显然“太郎具有尖锐的牙齿”和“太郎具有锋利的爪子”。根据规则 R1, 如果“太郎是哺乳动物”是明确的, 则“太郎是食肉动物”就是合理的。
(2) 根据数据“太郎身上有毛”和规则 R1, 显然“太郎是哺乳动物”。
4. 利用框架可以表示四种知识(典型的知识, 阶层的知识, 程序式的知识, 推理控制)。

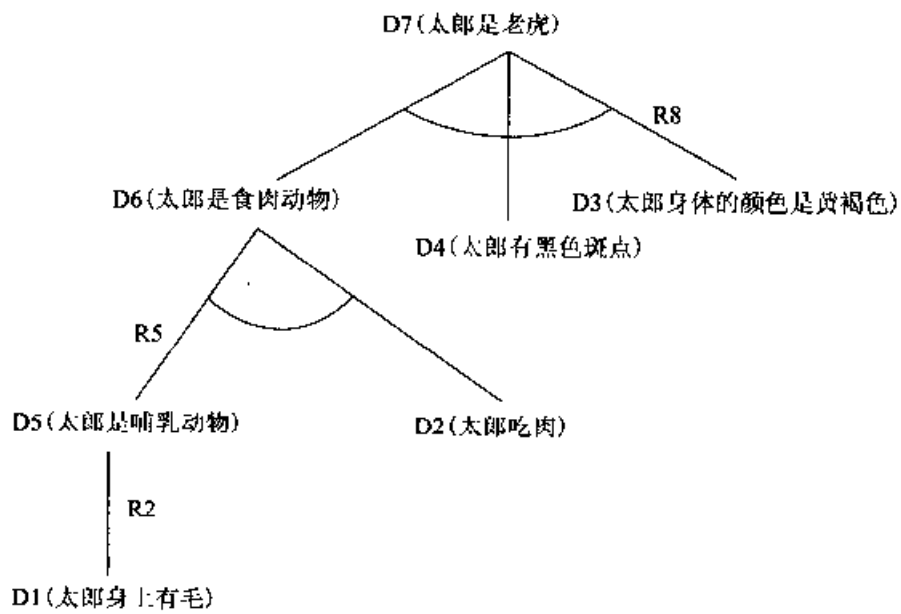


图 4

5. (1) 参看图 5。

会议38的框架	
时间	
值:	2000年10月2日
地点	
值:	信息会议室
目的	
值:	人工智能系统开发
出席者	
值:	佐藤・山田・铃木

图 5

(2) 参看图 6。

会议38的框架	
种类	
值:	AI会议
时间	
值:	2000年10月2日
地点	
值:	信息会议室

图 6

第4章

- 死记硬背式学习:地名地号的死记硬背,练习题答案的完全记忆等。
归纳式学习:坐在前面听讲的学生中,成绩好的居多。因此可以导出一条规则,这就是若坐在前面,则成绩会变好。
高效的学习:要阅读理解德语文章,但是手头只有德英辞典和英日辞典。这时,要经过德语→英语→日语这样两个步骤对辞典的查阅过程,对每次查阅的单词,以德日单词的条目登记成册。这种单词条目的制作,就是一种高效率的学习。
强化学习:在篮球的投篮比赛中,对球的情况的判断,发力的方法和角度的调节,均有赖于掌握投篮的窍门。
- 谓词逻辑表示:
优点:一般来说,因为表示能力强,所以适用于广泛的知识领域。
缺点:在一般的推理过程中,搜索空间过于广泛,不利于实用。
决策树表示:
优点:推理过程是机械的,可以得到实时解。
缺点:只能处理可以用决策树表示的知识。
- 略(参考表 4.2)。
- 关于原因,可以参阅课文,作为解决方法,例如可以考虑对各个宏规则(由高效学习得到的规则)的利用次数进行记录,并且清除利用率低的规则等。
- 利用 EBL 学习得到的“杯子”的概念是:

$$\text{杯子}(X) \leftarrow \text{平底}(X) \wedge \text{重量}(X, 0.5) \wedge \text{把手样式}(X) \wedge \text{向上呈凹形}(X),$$
 证明树省略。

第5章

- 参考式(5.1)和图 5.1 进行描绘。
- 变成与普通集合相同的集合。
- 参考式(5.5)和(5.7)进行描述。
- 请确认德摩根法则。
- 参考式(5.13)~式(5.15)和图 5.5,然后进行描绘。
- if 饱和时间长,并且透射率变成中等程度,then 洗涤时间加长。
- 参考洗衣机的例子求解该问题。
- 参考式(5.26)和图 5.8 进行求解。
- 参考式(5.34)、(5.35)、(5.37)与式(5.38)~(5.44)进行求解。
- 在 SGA 中,双亲个体被遗传操作产生的子个体置换,另外,选择操作也是随机进行的。
- 应用流动推销员等问题进行思考,就可以很好地进行理解。
- 以书末列举的文献为基础,考察 GA 的近期研究情况。

第 6 章

1. 由 $d(y, r^{(1)}) = d(y, r^{(2)})$, 得到

$$(y_1 - 2)^2 + (y_2 - 5)^2 = (y_1 - 6)^2 + (y_2 + 1)^2$$

$$y_1 - y_2 - 1 = 0$$

这是一条通过 $r^{(1)}$ 与 $r^{(2)}$ 的中点 $(4, 3)$, 并且与 $r^{(1)} - r^{(2)}$ 正交的直线。

2. 在一维的情况下, 根据设均值为 μ , 方差为 σ^2 , 式 (6.24) 变成

$$f(x) = \frac{1}{\sqrt{2\pi\sigma}} \exp\left[-\frac{1}{2\sigma^2}(x-\mu)^2\right]$$

把各个均值和方差代入上式, 并设 $x=9.0$, 得到

$$P(9.0|C_1) = \frac{1}{\sqrt{2\pi\sqrt{1}}} \exp\left[-\frac{1}{2 \cdot 1}(9-5)^2\right] = 0.000134$$

$$P(9.0|C_2) = \frac{1}{\sqrt{2\pi\sqrt{10}}} \exp\left[-\frac{1}{2 \cdot 10}(9-15)^2\right] = 0.020859$$

因为 $P(9.0|C_1) < P(9.0|C_2)$, 所以对于具有 $x=9.0$ 的特征模式的输入模式, 此时被识别为 C_2 类。

3. 若设两类为 C_1, C_2 , 在识别边界面上, 两类的均值向量的概率会变得相等, 因此两类的产生概率会相等, 即

$$P(c_1|x) = P(c_2|x)$$

$$P(x|c_1)P(c_1) = P(x|c_2)P(c_2)$$

因此得到

$$P(x|c_1) = P(x|c_2)$$

这个条件概率因为在 (6.24) 中已经给出, 设两个向量的均值向量为 m_1, m_2 , 若以相同的系数除之, 则得到

$$(x - m_1)' \sum^{-1} (x - m_1) = (x - m_2)' \sum^{-1} (x - m_2)$$

$$(m_1 - m_2)' \sum^{-1} x = \frac{1}{2} [m_1' \sum^{-1} m_1 - m_2' \sum^{-1} m_2]$$

$$(m_1 - m_2)' \sum^{-1} x = \frac{1}{2} [(m_1 - m_2)' \sum^{-1} (m_1 + m_2)]$$

式中若以下式代入

$$w = \sum^{-1} (m_1 - m_2)$$

$$x_0 = \frac{1}{2} (m_1 + m_2)$$

则识别边界面变成由下式描述的超平面:

$$w'(x - x_0) = 0$$

式中当协方差矩阵为单位矩阵时, 因为 $w (=m_1 - m_2)$ 与 $x - x_0$ 正交, 所以通过均值向量的中点, 显然构成了与均值向量连线正交的超平面。

4. 若依照算法顺序求 $g(i, j)$, 则得到表 1, 并且构成

$$g(I, J)/I = 5.0/6 = 0.83$$

线性伸缩的情况则变为

$$(|3-5| + |5-6| + |6-3| + |4-1| + |2-5|)$$

表 1 最小累积距离 $g(i, j)$

参 照 模 式	6	∞	∞	∞	7	9	5
	5	∞	∞	5	6	6	4
	1	∞	∞	8	5	4	8
	3	∞	4	5	3	4	6
	6	∞	3	2	4	8	8
	5	2	2	3	4	7	7
		3	5	6	4	2	5
		输入模式					

$$|(-5-6)|/6.0 = 2.17$$

5. 在输出符号 abb 时,被考虑的状态转移为 $S_1 S_1 S_2 S_1$ 和 $S_1 S_2 S_2 S_2$ 这两条路径。它们各自的概率 P_1 和 P_2 变为

$$P_1 = 0.25 \times 0.8 \times 0.75 \times 0.8 \times 0.5 \times 1.0 = 0.06$$

$$P_2 = 0.75 \times 0.2 \times 0.5 \times 0.4 \times 0.5 \times 1.0 = 0.015$$

模型输出 abb 的概率 $P(abb)$ 可以求得为

$$P(abb) = P_1 + P_2 = 0.06 + 0.015 = 0.075$$

另外,在彼得比算法的情况下,因为只考虑变成最大概率的状态转移序列,所以变成

$$P(abb) \approx P_1 = 0.06$$

第 7 章

1. 必要的特性可以设想,顺序为“符号性,阶层性,递归性,成长性”四种。关于每个特性的意义,可以参考 7.1 节。
2. 参考 7.1 节。
3. eval, quote, assert, retract, univ 都是一些用来提供从数据变换成程序,或者是从程序变换成数据功能的函数。eval 是把数据作为程序来执行的函数,quote 是把程序作为数据来处理的函数。另外,assert, retract 是在执行 Prolog 程序的过程中,动态地进行追加/清除的谓词,univ 是用来动态地编制程序的谓词。对于人工智能系统来说,伴随着系统的运行,处理程序自身的变化是很重要的(成长性),这些函数在实现这种变化中,具有最佳的功能。

4.

```
(defun second (list)
  (first (rest list))
)
(defun n-th (n list)
  (cond ((= n 1) (first list))
        (t (n-th (- n 1) (rest list))))
)
```

5.

```
second([X,Y|Z],Y).
```

```
n-th([X|Y],1,Y).
```

```
n-th([X|Y],N,Z); -NN is N-1,n-th(Y,NN,Z).
```

参 考 文 献

第1章

- [1] ダグラス・R. ホフスタッター (野崎昭弘, はやしはじめ, 柳瀬尚紀訳) : ゲーデル, エッシャー, バッハ, 白揚社 (1985).
- [2] Newell, A. and Simon, H. A. : GPS, A Program that Simulates Human Thought, Computers and Thought, E. A. Feigenbaum and J. Feldman Eds., pp.297-293 (1963)
- [3] Russell, S. and Norvig, P. : Artificial Intelligence : A Modern Approach, Prentice, Hall (1995) (古川康一 監訳 : 人工知能・エージェントアプローチ, 共立出版, (1997)).

第2章

- [1] Pearl, J. : Heuristics : Intelligent Search Strategies for Computer Problem Solving, Addison Wesley Publishing Company (1984) (ヒューリスティック探索のみを扱ったよくまとめられた教科書)
- [2] Banerji, R. B. : Artificial Intelligence : A Theoretical Approach, Elsevier North Holland, Inc. (1980) 高原康彦, 中野文平, 宇治橋義弘 訳 : 人工知能 : コンピュータによるゲーム, 共立出版 (1983) (探索を用いた問題解決手法を理論的に議論した本)
- [3] Bolc, L. and Cytowski, J. : Search Methods for Artificial Intelligence, Academic Press Inc. (1992) (ヒューリスティック探索の教科書, C言語によるアルゴリズムの記述がなされている)
(そのほか一般的な人工知能の教科書にも含まれている)
- [4] Nilsson, N. J. : Principles of Artificial Intelligence, Tioga Publishing Co. (1980) 白井良明, 辻井潤一, 佐藤泰介 訳 : 人工知能の原理, 日本コンピュータ協会 (1983)
- [5] Charniak, E. and McDermott, D. : Introduction to Artificial Intelligence (1986)
- [6] Rich, E. and Knight, K. : Artificial Intelligence, 2nd Edition, McGraw-Hill, Inc (1991)
- [7] Winston, P. H. : Artificial Intelligence, 3rd Edition, Addison Wesley Publishing Company (1992)
- [8] Ginsberg, M. : Essentials of Artificial Intelligence, Morgan Kaufmann Publishers (1993)
- [9] Russell, S. J. and Norvig, P. : Artificial Intelligence : A Modern Approach, Prentice Hall (1995)
- [10] 白井良明, 辻井潤一 : 人工知能, 岩波書店 (1982)
- [11] 太原育夫 : 人工知能の基礎知識, 近代科学社 (1988)
- [12] 長尾真 : 知識と推論, 岩波書店 (1988)

第3章

- [1] 北橋忠宏 : 知識情報処理, 森北出版 (1998)
- [2] Russell, S. and Norvig, P. (古川康一 監訳) : エージェントアプローチ人工知能, 共立出版 (1997)

- [3] 溝口理一郎：エキスパートシステムI, II, III, 朝倉書店 (1993)
- [4] 溝口理一郎, 池田満：オントロジー工学序説－内容指向研究の基盤技術と理論の確立を目指して－, 人工知能学会誌, 12, 4, pp.559-569 (1997)
- [5] 情報処理学会編：知識工学, オーム社 (1987)

第4章

- [1] 荒屋眞二：人工知能概論, 共立出版 (1992)
- [2] 太原育夫：人工知能の基礎知識, 近代科学社 (1988)
- [3] 志村正道：人工知能, 森北出版 (1994)
- [4] 廣田薫編著：知能工学概論, 昭晃堂 (1996)
- [5] 淵一博 監修, 古川康一, 溝口文雄共編：知識情報処理シリーズ, 第2巻, 知識の学習メカニズム, 共立出版 (1986)
- [6] Quinlan, J. R. : C4.5 : Programs for Machine Learning, Morgan Kaufmann Publishers (1993)
- [7] Winston, P. H. : Artificial Intelligence (3rd Edition), Addison Wesley (1992)
- [8] Winston, P.H.編, 白井良明, 杉原厚吉訳：コンピュータビジョンの心理 (The Psychology of Computer Vision), 産業図書 (1979)
- [9] Cohen, P. R. and Feigenbaum, E. A. 編, 田中幸吉, 淵一博監訳：人工知能ハンドブック, 第III巻, 共立出版 (1984)
- [10] Russell, S. and Norvig, P. : Artificial Intelligence : A Modern Approach, Prentice Hall International (1995)
- [11] 特集：「学習と知識獲得技術の新展開」, 人工知能学会誌, 3, 6 (1988)
- [12] パネル討論：「機械学習の理論と実際」, 人工知能学会誌, 7, 1 (1992)
- [13] 小特集：「最近の機械学習」, 人工知能学会誌, 9, 6 (1994)
- [14] 特集：「強化学習」, 人工知能学会誌, 12, 6 (1997)
- [15] 日本認知科学会：認知科学の展望, 第4巻, 講談社 (1991)

第5章

- [1] 菅野道夫, 向殿政男監訳：ザデー・ファジィ理論, 日刊工業新聞社 (1992)
- [2] 坂和正敏：ファジィ理論の基礎と応用, 森北出版 (1989)
- [3] 田中英夫：ファジィモデリングとその応用, 朝倉書店 (1990)
- [4] 日本ファジィ学会編：講座ファジィ5－ファジィ制御－, 日刊工業新聞社 (1993)
- [5] 長町三生編：ファジィ化製品開発の基礎と実際, 海文堂 (1991)
- [6] 合原一幸：ニューラルコンピュータ, 東京電機大学出版局 (1988)
- [7] 八木和夫, 鈴木義武：ニューロ情報処理技術, 海文堂 (1992)
- [8] 萩原将文：ニューロ・ファジィ・遺伝的アルゴリズム, 産業図書 (1994)
- [9] 甘利俊一：神経回路網の数理, 産業図書 (1978)
- [10] 合原一幸 編著：ニューロ・ファジィ・カオス, オーム社 (1993)
- [11] 甘利俊一, 向殿政男：ニューロとファジィ, 培風館 (1994)
- [12] 林 勲, 占橋 武 編著：ファジィ・ニューラルネットワーク, 朝倉書店 (1996)
- [13] 木村資生：生物進化を考える, 岩波新書19, 岩波書店 (1988)

- [14] 中原, 佐川: 進化論が変わる—ダーウィンをゆるがす分子生物学, ブルーバックス, 講談社 (1991)
- [15] 坂和, 田中: 遺伝的アルゴリズム, 日本ファジィ学会編, 朝倉書店 (1995)
- [16] 伊庭斉志: 遺伝的アルゴリズムの基礎—GAの謎を解く—, オーム社 (1994)
- [17] 北野宏明編著: 遺伝的アルゴリズム, 産業図書 (1993)
- [18] 北野宏明編著: 遺伝的アルゴリズム2, 産業図書 (1995)
- [19] 北野宏明編著: 遺伝的アルゴリズム3, 産業図書 (1997)
- [20] 安居院, 長尾: ジェネティックアルゴリズム, 昭晃堂 (1993)
- [21] Goldberg, D. E.: Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley Publishing Company, Inc. (1989).
- [22] Fogel, D. B.: Evolutionary Computation, IEEE Press (1995).
- [23] Schwefel, H.-P.: Evolution and Optimum Seeking, John Wiley & Sons (1995).
- [24] Baeck, T.: Evolutionary Algorithms in Theory and Practice, Oxford (1996).
- [25] 日本ファジィ学会 <http://soft.amcac.ac.jp/>
- [26] 日本神経回路学会 <http://jnns.inf.eng.tamagawa.ac.jp/>
- [27] Genetic Algorithms Archive <http://www.aic.nrl.navy.mil/galist/> (米国The Navy Center for Applied Research in Artificial Intelligenceから発信されているGA並びに進化的計算のニュース, 学会カレンダー, 関連リンク集などが網羅されている)

第6章

- [1] 斎藤収三, 中田和男: 音声情報処理の基礎, オーム社 (1981)
- [2] 中田和男: 音声, 音響工学講座7, コロナ社 (1995)
- [3] 古井貞熙: デジタル音声処理, 東海大学出版会 (1985)
- [4] 今井 聖: 音声認識, 共立出版 (1995)
- [5] 中川聖一: 確率モデルによる音声認識, 電子情報通信学会 (1988)
- [6] 赤穂昭太郎: EMアルゴリズムの幾何学, 情報処理, 37, 1, pp.43-51 (1996)
- [7] 北 研二, 中村哲, 永田昌明: 音声言語処理—コーパスに基づくアプローチ—, 森北出版 (1996)

第7章

- [1] 後藤滋樹: 記号処理プログラミング, 岩波講座ソフトウェア科学第8巻, 岩波書店 (1988)
- [2] ウィンストン, ホーン (白井, 安部, 井田訳): Lisp I, II, 培風館情報処理シリーズ 4-2, 培風館 (1991)
- [3] 安西祐一郎, 佐伯 胖, 無藤 隆: Lispで学ぶ認知心理学I 学習, 東京大学出版会 (1981)
- [4] 黒川利明: LISP入門 (電子計算機のプログラミング7), 培風館 (1982)
- [5] 竹内郁雄: 初めての人のためのLISP, サイエンス社 (1986)
- [6] 衣笠成一: 実用Common Lisp, 日刊工業新聞社 (1987)
- [7] 湯浅太一, 萩谷昌巳: Common Lisp入門, 岩波書店 (1986)
- [8] 湯浅太一: Scheme入門, 岩波書店 (1991)
- [9] 井田哲雄: 計算モデルの基礎理論, 岩波講座ソフトウェア科学, 第12巻, 岩波書店 (1991)
- [10] コワルスキ (山田, 菊池, 桑野訳): 論理による問題の解法—Prolog入門— (情報処理

- シリーズ8), 培風館 (1987)
- [11] 長尾 真, 淵 一博: 論理と意味, 岩波講座情報科学7, 岩波書店 (1983)
- [12] ラマンドラン・バラス (斉藤, 舟木訳): Prolog詳説—対話形式によるアプローチャー, 啓学出版 (1990)
- [13] Bratko, I. (安部 訳): Prologへの入門I—PrologとAI—, 近代科学社 (1990)
- [14] Bratko, I. (安部, 田中 訳): Prologへの入門2—AIプログラミング—, 近代科学社 (1996)
- [15] Clocksin, W. F. and Mellish, C. S.: Programming in PROLOG, 3rd Revised and Extended ed., Springer-Verlag (1987)
- [16] ロイド (佐藤, 森下 訳): 論理プログラミングの基礎, ソフトウェアサイエンスシリーズ, 産業図書 (1987)

- シリーズ8), 培風館 (1987)
- [11] 長尾 真, 淵 一博: 論理と意味, 岩波講座情報科学7, 岩波書店 (1983)
- [12] ラマンドラン・バラス (斉藤, 舟木訳): Prolog詳説—対話形式によるアプローチャー, 啓学出版 (1990)
- [13] Bratko, I. (安部 訳): Prologへの入門I—PrologとAI—, 近代科学社 (1990)
- [14] Bratko, I. (安部, 田中 訳): Prologへの入門2—AIプログラミング—, 近代科学社 (1996)
- [15] Clocksin, W. F. and Mellish, C. S.: Programming in PROLOG, 3rd Revised and Extended ed., Springer-Verlag (1987)
- [16] ロイド (佐藤, 森下 訳): 論理プログラミングの基礎, ソフトウェアサイエンスシリーズ, 産業図書 (1987)

- シリーズ8), 培風館 (1987)
- [11] 長尾 真, 淵 一博: 論理と意味, 岩波講座情報科学7, 岩波書店 (1983)
- [12] ラマンドラン・バラス (斉藤, 舟木訳): Prolog詳説—対話形式によるアプローチャー, 啓学出版 (1990)
- [13] Bratko, I. (安部 訳): Prologへの入門I—PrologとAI—, 近代科学社 (1990)
- [14] Bratko, I. (安部, 田中 訳): Prologへの入門2—AIプログラミング—, 近代科学社 (1996)
- [15] Clocksin, W. F. and Mellish, C. S.: Programming in PROLOG, 3rd Revised and Extended ed., Springer-Verlag (1987)
- [16] ロイド (佐藤, 森下 訳): 論理プログラミングの基礎, ソフトウェアサイエンスシリーズ, 産業図書 (1987)