COMPUTATIONAL INTELLIGENCE FOR OPTIMIZATION

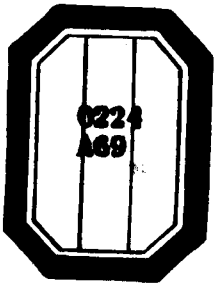
用于最优化的计算智能

Nirwan Ansari Edwin Hou 著
李 军 边肇祺 译

清华大学出版社
http://www.tup.tsinghua.edu.cn

用于最优化的计算智能

Nirwan Ansari Edwin Hou 著
李军　边肇祺　译



清华大学出版社

(京)新登字 158 号

内 容 简 介

本书从讨论组合优化中的基本问题——NP 问题入手，系统地讲述了近年来所发展起来的智能最优化的各种技术和方法，其中包括启发式搜索、Hopfield 神经网络、模拟退火和随机机、均场退火以及遗传算法等；并在此基础上，通过一些典型的应用问题，如旅行商问题、模式识别中的点模式匹配问题、通信和任务调度等问题进一步阐明以上一些基本方法怎样用来解决这些原来具有 NP 性质的困难问题。本书是作者在美国新泽西州理工学院多年讲授有关课程的基础上写成的。全书深入浅出，理论联系实际。为帮助学生掌握基本概念，提高学习能动性，各章编写了习题。

本书可作为通信、计算机、控制各专业的高年级学生和研究生学习有关课程的教材。它对于广大科研工作者也是一本很有实际价值的参考书。

Computational Intelligence for Optimization

Nirwan Ansari Edwin Hou

北京市版权局著作权合同登记号：01-98-0945

书　名：用于最优化的计算智能

作　者：Nirwan Ansari Edwin Hou 著　李军　边肇祺　译

出版者：清华大学出版社(北京清华大学学研楼，邮编 100084)

http://www.tup.tsinghua.edu.cn

印刷者：清华大学印刷厂印刷

发行者：新华书店总店北京发行所

开　本：787×1092 1/16　**印张**：8.75　**字数**：204 千字

版　次：1999 年 12 月第 1 版　　1999 年 12 月第 1 次印刷

书　号：ISBN 7-302-03635-7/TP・2019

印　数：0001～3000

定　价：18.00 元

译者的话

洪伟年(Nirwan Ansari)与侯瑞海(Edwin Hou)两位教授集多年教学和科研成果合著的《用于最优化的计算智能》一书,是反映近年来最优化技术发展的专著。此书取材新颖、内容丰富、结构合理、论述清楚,在有限的篇幅中对所有重要方法都进行了透彻的阐述,不仅对于研究生掌握有关概念和方法十分有益,对通信、计算机、控制等领域的科研和工程技术人员也很有参考价值。

最近数年来计算智能技术发展迅速,在此基础上,本书论述了最优化基础理论和实际应用。书中涉及的多种智能最优化方法均已成为人工智能、模式识别和其它有关领域行之有效的方法,并已与经典的统计模式识别技术和结构模式识别技术一起,构成了现代模式识别方法的理论基础。鉴于国内有关智能计算在最优化中应用的资料大多散见于各种学术期刊和会议文集,相信本书中译版的出版将会有助于这一学科的发展,并推动相关领域的教学、科研和工程实践。

为了便于读者根据不同的学习和工作需要较快地掌握有关的算法,著者采用了一种相对独立的模块式结构。书中第 2～6 章每章各介绍一种基本的方法,首先对有关概念给出严格而明确的定义,然后再深入浅出地展开对理论部分的论述。在第 7～11 章每章围绕一个典型问题,对各种方法的应用实例加以描述,并进行适当比较。这样的安排不仅给读者一个清晰的概念,而且引导读者对各种方法的特点进行深入的比较,从而学会针对实际问题的特点选择有效的解决方法。

本书从第 1 章到第 8 章以及第 10 章,第 11 章由李军翻译,第 9 章由边肇祺翻译,边肇祺并对全书进行了校正。书中难免有翻译不确切的地方,望请读者批评指正。

感谢两位作者在本书的翻译过程中给予的帮助。清华大学出版社为本书的出版做了大量工作,译者在此一并致谢。

译　者

1998 年 11 月

前　言

最优化在本质上是一门交叉学科。它对多个学科产生了重大影响，并已成为不同领域中很多工作都不可或缺的工具。传统最优化方法已经十分完善且有多部优秀教材出版，然而一些已被证明在解决全局最优化问题上行之有效的新方法又涌现了出来，例如神经网络、模拟退火、随机机、均场理论和遗传算法等。

本书为高级最优化技术的最新发展，特别是启发式搜索、神经网络、模拟退火、随机机、均场理论和遗传算法，提供一个技术性的描述，并着重强调它们的数学基础、实现方法以及实际应用。这一教材适用于电类、计算机类等专业一年级研究生的教学，也可作为工程技术人员、科研人员及其他专业人员的参考书。本书是过去五年中我们所授几门课程的结晶，也凝聚了我们在这一领域多项交叉学科研究的成果。在过去的十年里，上述高级最优化技术受到越来越多的重视，但是新书却很少出版。绝大多数新的最优化理论及其应用都散见于期刊杂志、技术报告和会议文集。这使得初学入门者难于进行系统学习。我们希望这本书能使读者对这些方法综合了解，从而填补科学文献上的这一空缺。

本书在内容安排上采用了每一章都相对独立的模块式结构，其中第 2～6 章每章覆盖了一项最优化技术的理论，而第 7～11 章则侧重于不同范畴里最优化技术的实际应用。每章既可单独学习，又可作为一个大的综合性题目来研究。

我们深深得益于新泽西州理工学院那些在过去的五年中学习了我们的研究生课程“高级最优化技术”和“人工神经网络”的研究生们，尤其是 G. Wang，J. Chen，D. Liu，A. Agrawal，Y. Yu，Z. Zhang，A. Arulumbalam，S. Balasekar，J. Li 和 N. Sezgin。他们这些年来的反馈和评论帮助我们形成了本书现在的样式。我们非常感谢 L. Fitton 对于编辑此书的帮助以及 Kluwer 学术出版社的 S. Rumsey 在手稿输入方面所提供的支持。最后，我们衷心地表达对 Kluwer 学术出版社 A. Greene 的谢忱，感谢他对我们完成此书的不断鼓励。

Nirwan Ansari
Edwin Hou

目　录

第1章 引 言

许多科学和工程问题都可以归结为有约束条件的最优化问题，并可用如下数学公式加以表达：

$$\min_{x\in S} f(x) \text{ 基于 } g(x) \tag{1.1}$$

其中 S 是解域，f 是价值函数，g 是约束条件集合。各种最优化问题可以根据 S，f 和 g 的特点加以分类。如果 f 和 g 都是线性函数，则(1.1)式所描述的是一个易于求解的线性最优化问题。否则，(1.1)式便成为较难解决的非线性最优化问题。线性规划是一类典型的线性最优化问题，其约束条件的形式为 $g(x)\geqslant 0$ 或 $g(x)=0$。线性规划问题可以用很简单的算法[125]求解，并能在有限步骤内求得最优解。然而，在工程和其它领域中遇到的问题有很多都属于“难于求解”的一类问题，无法使用确定性算法求解。例如旅行商问题(TSP)和各种调度问题等。

包括神经网络、模拟退火、随机机、均场理论和遗传算法在内的各种最优化新方法的发现和发展，使得更有效地求解那些可以良好定义的难题成为可能。本书就以这些新的最优化方法和它们的应用为主题。

1.1 计算复杂度

在很多情况下，一个最优化问题可以用许多方法加以解决，而每种方法又能够采取多种算法予以实现。由于所有这些算法全都可能给出同一最优解，人们需要根据求得最优解的效率对它们进行分类。当然，一个最优化问题也有可能根本难以解决，因而不存在有效的寻求最优解的方法。计算复杂度理论就是用于研究算法有效性和问题难度的手段[57][94][110]。这一节主要介绍有关计算复杂度理论的一些基本概念。

1.1.1 算法分析

为便于有效地对算法进行真实比较，我们假定编译效率、程序语言以及计算机运算速度等因素对于有关算法来说都是完全相同的。

算法所用时间或空间随问题变大的增长率是衡量算法有效性的一个重要尺度。问题的“大小”是以输入数据量来计算的，例如旅行商问题中城市的数目。算法占用的时间和空间，亦即算法的时间复杂度和空间复杂度，则取决于算法用于求解问题所需的计算机运行时间和存储空间。研究一个算法随问题的规模增长时的极限性能，即渐进时间/空间复杂度，也是很有意义的。

我们可以定义一个以数据量为输入，时间/空间占据量为输出的函数，用来度量算法的量化时间/空间复杂度。例如，需要求解的问题是将一组整数由小到大排序。解决手段

之一是利用冒泡排序算法[10]比较相邻两数并进行适当交换。冒泡排序算法在最坏情况下的时间复杂度可以表示为

$$f(n)=\frac{n(n-1)}{2}=\frac{n^2}{2}-\frac{n}{2} \tag{1.2}$$

其中 n 是被排序整数的个数。

下表所列正是冒泡排序算法在不同输入数据量下的时间复杂度，它显示出每当输入量增加一个数量级时，冒泡排序算法的时间复杂度大致增加两个数量级。换句话说，时间复杂度以 n 的二次指数增长，对应于(1.2)式函数中的二次项 $n^2/2$。

n	1	10	100	1000	10000	10^5
$f(n)$	0	45	4950	499500	5.0×10^7	5.0×10^9

由于算法的渐进复杂度是我们关注的对象，余项 $-n/2$ 可以忽略不计，而二次项中的常数因子 1/2 也可省去，因此，冒泡排序算法的时间复杂度为 n^2 数量级，或记为 $O(n^2)$。

定义 1.1 若 f 和 g 是定义在正整数域上的正函数，$f,g:I^+\to R^+$，则

(1) $f=O(g)$，如果存在正常数 c 和 N，对所有的 $n\geqslant N$ 都使得 $f(n)\leqslant c\cdot g(n)$。

(2) $f=\Omega(g)$，如果存在正常数 c 和 N，对所有的 $n\geqslant N$ 都使得 $f(n)\geqslant c\cdot g(n)$。

(3) $f=\Theta(g)$，如果存在正常数 c,d 和 N，对所有的 $n\geqslant N$ 都使得 $d\cdot g(n)\leqslant f(n)\leqslant c\cdot g(n)$。

例如，(1.2)式中有

$$\frac{n(n-1)}{2}\leqslant n^2,\text{其中 } n\geqslant 1$$

因此可有 $c=1$ 和 $N=1$，使得 $f=O(n^2)$。

在很多情况下，算法的复杂度为下列函数之一：

$$O(\log n),O(n),O(n\log n),O(n^2),O(2^n),O(n^{\log n}),O(n!),O(n^n)$$

要注意的是，上述函数是按复杂度增大排列的。

表 1.1 时间复杂度函数的比较

时间复杂度函数	输入量 n				
	10	20	30	40	100
n	10 ns	20 ns	30 ns	40 ns	100 ns
$n\log n$	10 ns	26.0 ns	44.3 ns	64.1 ns	200 ns
n^2	100 ns	400 ns	900 ns	1.6 μs	10 μs
2^n	1.0 μs	1.0 ms	1.1 s	18.3 min	4.0 世纪
$n!$	3.6 ms	77.1 年	8.4×10^{13} 世纪	2.6×10^{29} 世纪	3.0×10^{139} 世纪

表 1.1 在假定所用计算机每秒可以执行 10 亿次运算的前提下，比较了几个方程在不

同输入量条件下的时间复杂度。很明显，如果一个算法的复杂度是指数函数（例如 2^n 或 $n!$），它在大输入量情况下是无望求解的。当一个算法的时间复杂度方程可以用多项式方程来划界时，我们称之为多项式时间算法；否则，它是一个指数时间算法。

1.1.2 NP 完全问题

如上所述，多项式时间算法比指数时间算法易解得多。不过总是有一些最优化问题无法用多项式时间算法求解。这些问题被称为难解问题。事先了解一个问题是否为难解问题自然会有很多好处。NP 完全理论的建立就是为了确定问题的难解度，并检验各个问题在难易程度上的相互关系。这一 NP 完全理论的基础是由库克(Cook)[46]确立的。

(1) 如果一个问题 P 可以被一个多项式时间算法求解，而另一个问题 Q 可以在多项式时间内被变换为 P，则一定存在一个多项式时间算法可以求解 Q。其中在多项式时间内将一个问题转换到另一个问题的过程称为多项式时间约化。

(2) 决策问题(答案为“是”或“否”的一类问题)中可在多项式时间内用非确定性计算机① 解出的特定类型被称为 NP。

(3) NP 中的任一问题都可以被多项式约化为 NP 中的一个特定决策问题，即满意度问题。因此，如果可以找到一个多项式时间复杂度算法来解决这一满意度问题，则 NP 中的任一问题都可在多项式时间内得解。另一方面，如果 NP 中任一问题是难解问题，则此一满意度问题也是难解问题。满意度问题可被认为是 NP 中“最难”的问题。

(4) 存在一组使 NP 中的其它问题以多项式方式约化于它们的“最难”问题。

有很多优化算法，诸如旅行商问题、多处理器任务调度问题等，都被发现可以多项式约化为 NP 中最难的问题。这些问题构成了 NP 完全问题类。在文献[57]中可以查到它们。究竟 NP 完全问题是否难解仍然是个谜。虽然很多线索表明 NP 完全问题确为难解，但至今未见严格的证明或证伪。不过，如果可以表明一个问题属于 NP 完全问题类，它多半是难于求解的。

1.2 最优化技术综述

这一节简要介绍各种最近发展起来的最优化新方法。

1.2.1 启发式搜索

启发式搜索最初是作为人工智能中问题求解程序的搜索器而被开发出来的[117]。在多数情况下，待解问题可以用状态空间加以表达，并用状态空间或图搜索算法求解。宽度优先和深度优先等图搜索算法的应用范围有限。不过，与待解问题相关的(启发)信息一般来说是可获取并运用于搜索过程。这使得最佳优先算法通常会有较好的表现。A* 算法[73]适于求解状态空间中从起始节点到终止节点的最小代价路径。它依据代价函数来选择下一个被搜索或扩展的节点。代价函数计算出从起始节点到当前节点路径的代价，并用启发知

① 非确定性计算机是一类计算模型，例如非确定性图灵(Turing)机。

识估计出到终止节点的余下路径的代价。在满足“可容性”的条件下，如果存在最小代价路径，A* 一定可以找到。

1.2.2 霍普费尔德神经网络

自从霍普费尔德(Hopfield)揭示了他所提出的计算网络的运算能力[84]以来，大量的工作被投入霍普费尔德神经网络(简称霍氏网)的分析和应用。虽然最初的网络结构有它的弱点，霍普费尔德开创性的成果仍旧在某种程度上打通了一条神经网络的复兴之路。霍氏网的收敛性和稳定性已经被证明。加以适当改进后，霍氏网可用于求解旅行商问题等许多有约束条件的最优化问题。

将一个有约束条件的最优化问题抽象为可由霍氏网解决的形式，需要

(1) 将代价函数和约束条件转换为霍普费尔德能量函数；

(2) 确定拉格朗日(Lagrange)参数。

上述步骤往往是成功应用霍氏网的关键。许多研究人员未能复现霍普费尔德求解旅行商问题的结果，毛病经常就出在这里。在后续章节中，我们将讨论霍氏网的稳定性及其应用。

1.2.3 模拟退火与随机机

模拟退火算法最初由科克派特里克(Kirkpatrick)、小哥拉特(Gelatt Jr.)和瓦克奇(Vecchi)发明[98]。它模仿了液体结晶的过程：

(1) 在高温下，粒子能量较高，可以自由运动和重新排序；

(2) 随着温度的下降，粒子能量减弱，运动减少；

(3) 粒子最终进入平衡态，固化为具有最小能量的晶体。

模拟退火算法需要两个主要操作：一个是热静力学操作，用于安排降温过程；另一个是随机张弛操作，用于搜索在特定温度下的平衡态。模拟退火算法的长处在于它具有跳离局部最优解的能力。在给定温度下，模拟退火算法不但进行局部搜索，而且可以在一定概率下“爬山”到代价更高的解答以防止搜索进程陷入局部最小解。应用模拟退火算法以及各种不同的统计特性于霍氏网，可以得到波尔兹曼(Boltzmann)、高斯(Gaussian)、柯西(Cauchy)等随机机。类似于霍氏网的是，能量函数的拉格朗日参数对随机机的性能有重要影响。

1.2.4 均场退火

为了能以比随机机中的随机张弛操作更快的速度达到热平衡态，神经元的值可以由它们的平均值来加以近似。这时，在每一温度下，对神经元施加的是一个确定性的而不是随机性的操作。与随机机相比，这一网络只需较少的迭代次数便可在每一温度下达到热平衡态。在退火过程中，这一近似方法被称为均场退火。它在性能和计算复杂度二者之间作了一个很好的折衷。

1.2.5 遗传算法

遗传算法是一种模拟自然选择和遗传的随机搜索算法。它由贺兰德(Holland)提出[83],最初用于研究自然系统的适应过程和设计具有自适应性能的软件。遗传算法的基本形式要求有一个染色体表示用来对参数空间编码、一个适度函数用于估价所有的染色体、一组遗传操作用于产生新的染色体和一组概率用于控制遗传操作。遗传算法已被非常成功地应用于解决许多最优化问题并越来越流行。

1.3 本书的结构

本书分为两部分,其中第2～6章介绍5种先进优化算法的背景和理论,第7～14章描述这些算法在不同工程领域里的实际应用。具体而言,第2章讨论以A*算法为主的启发式搜索方法,包括图表示、图搜索算法、启发式算法、A*算法和可容性。第3章描述霍普费尔德神经网络,并讨论如何将一个有约束条件的最优化问题转化为一个可用霍氏网求解的无约束条件的最优化问题。第4章从最初的麦绰泼里斯(Metropolis)算法入手,沿着模拟退火算法的发展途径叙述了用模拟退火取代霍氏网中的确定性规则从而推导出随机机的各种方法。第5章描述如何利用神经网络中神经元的热均值来近似模拟退火的随机过程。这一称为均场退火的近似方法提供了寻找最优解的有效方法。第6章深入探讨遗传算法,包括遗传算法的简介、遗传操作及其分析等。第7章讨论旅行商问题,并给出分别用启发式搜索、模拟退火算法、霍氏网和遗传算法求解的方法。第8章以均场退火在卫星广播调度和集成TDMA通信系统数据吞吐量最大化中的应用为例,描述优化算法在远程通信中的应用。第9章讨论计算机视觉中十分重要的点模式匹配问题,并给出基于模拟退火和遗传算法的求解方法。第10章讨论多处理机系统中的任务调度问题及其基于均场退火和遗传算法的两种不同求解方法。第11章讨论制造系统中的作业调度问题。这一章还要讨论这一问题基于遗传算法的求解方法,并与传统方法加以比较。

本书采用模块式结构,第一部分中每章都可单独学习,然后可从第二部分有关应用的章节中获取更多的技术细节。

1.4 习题

1.1 证明:若 $f_1=O(g_1)$ 且 $f_2=O(g_2)$,则

$$f_1 + f_2 = O(g_1 + g_2)。$$

1.2 证明:若

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = a,\ 0 < a < \infty,$$

则 $f=O(g)$。

第2章　启发式搜索方法

启发式搜索方法依靠任务无关信息来简化搜索进程。在很多情况下，问题求解可视为系统化地构造或查找解答的过程。因此，在人工智能领域中，开发出了用于计算机问题求解的各种不同搜索算法[117]。尽管在某些情况下并非十分有效，迷宫、定理证明、国际象棋等问题都可以由计算机程序给出解答。一般来说，搜索过程包括检查搜索空间、估价可能有解的各种不同路径、记录已经搜索到的各个路径。

为简化搜索并减少搜索过程中时有出现的大量可选路径，从与待解问题有关的信息中所得到的启发知识或“经验法则”可用来确定搜索的方向。精心选择的启发信息可在很大程度上加大找出解答的机会，且在同时减少求解所需的时间。这一章统观启发式搜索方法并着重介绍 A* 算法。

在如图 2.1 所示的 3×3 九宫格棋盘上，摆放着 8 个将牌，上面不重复地刻着从 1 到 8 这 8 个数码。棋盘上余下一个空格使得其相邻将牌可以移动到它上面。设法找到一个将牌移动序列，用最少次数的移动将给定的起始布局转换为给定的终止布局，这就是 8 数码问题。实际上，记录空格的移动比记录将牌的移动方便得多。

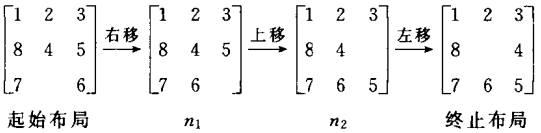


图 2.1　8 数码问题

在求取最小移动次数时，可用对各个布局的评价作为代价函数，从而把 8 数码问题变成一个最小代价问题。代价函数可被分为两项：一项是至今已经付出的代价；另一项则是基于启发信息估计出的到达目标尚需付出的代价。在代价函数中引入适当选择的启发信息（见 2.2 节），计算机搜索程序能够有效地解决 8 数码问题。

2.1　图搜索算法

图是构造搜索空间、开展搜索过程的便利工具。

定义 2.1　图 $G=(V,E)$ 是一个二元组，其中 $V=\{v_1,v_2,\cdots,v_n\}$ 是节点的有限集，$E=\{(v_i,v_j)\subseteq V\times V\}$ 是连结 V 中节点的连线集。

一个图可为有向图亦可为无向图。有向图的连线是有方向性的，因而从 v_i 到 v_j 的连线 (v_i,v_j) 与从 v_j 到 v_i 的连线 (v_j,v_i) 是不同的。但在无向图中，连线 (v_i,v_j) 与 (v_j,v_i) 是相同的。图 2.2 是有向图 $G=(\{v_1,v_2,v_3,v_4,v_5\},\{(v_1,v_2),(v_1,v_4),(v_2,v_5),(v_3,v_2),(v_3,v_4),(v_4,v_3)\})$ 的图示。

定义 2.2 在有向图 $G=(V,E)$ 中，若 $v_i,v_j\in V$ 且 $(v_i,v_j)\in E$，亦即存在连线 (v_i,v_j)，则节点 v_j 是节点 v_i 的子节点或称后辈节点。同样，v_i 是 v_j 的父节点或称前辈节点。

定义 2.3 树是一类除根节点外的每个节点都只有一个父节点的图。

如果我们从图 2.2 所示有向图中去掉两根连线 (v_3,v_2) 和 (v_3,v_4)，则所余下的就是一个树。在搜索过程中，树被用来记录已展开的路径。

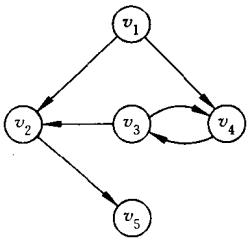


图 2.2 有向图示例

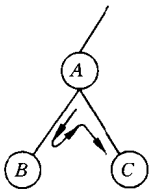


图 2.3 回溯示例

多数情况下，有待考察的图是一个有向图，通常还有唯一的一个起始节点。搜索过程不断生成后辈节点并把它们加入图中。这一过程被称为节点展开。搜索持续进行，直到它找到属于终止节点集的任一节点。以图的形式记录搜索过程的一个重要原因是为了便于回溯。回溯使得搜索过程得以废弃任何失败的路径，从而重新开拓一条崭新的路径。以图 2.3 所示搜索图为例，如果搜索过程到达节点 B 但它却似乎无助于求解，则搜索过程可以回溯到父节点 A 并从节点 C 继续。

一个路径是否有助于求解通常决定于代价函数。它对一个节点的评价基于至今已经花费的代价和其通往目标节点的可能性。代价函数可与图中的连线一起被用来表达搜索过程中从一个节点到另一个节点的代价。

定义 2.4 $cost(v_i,v_j)$ 是节点 v_i 与节点 v_j 间连线的代价。

定义 2.5 节点序列 $v_1\rightarrow v_2\rightarrow\cdots\rightarrow v_k$ 是从节点 v_1 到节点 v_k 的长度为 k 的路径。其中对于 $j=1,2,\cdots,k-1$，有 v_{j+1} 为 v_j 的子节点。

一条路径的代价也就是从起始节点经由如下公式列出的中间节点到达目标节点的路径上所有连线代价之和，即

$$cost(n_1\rightarrow\cdots\rightarrow n_k)=\sum_{j=1}^{k-1}cost(n_j,n_{j+1}) \tag{2.1}$$

因此，要评价节点 n，我们可以建立一个代价函数 $f(n)$，并使其包含两个组成部分：其一是从起始节点到 n 的实际代价；其二是从 n 到终止节点的代价估值。下面给出的基于 $f(n)$ 的图搜索过程 Graph-Search 试图找到一个从起始节点到终止节点的最小代价路径。

Graph-Search 算法

输入：起始节点 s 和一组终止节点 $\{t_i\}$。

输出：图 G 和树 T。

(1) 初始化。以起始节点 s 构造图 G 和集合 $OPEN$，$G\leftarrow\{s\}$，$OPEN\leftarrow\{s\}$，且令集合

CLOSE 为空集。

(2) LOOP：若 *OPEN* 为空集，算法以失败结束。

(3) 从 *OPEN* 中取出第一个节点 n，并使 $OPEN \leftarrow OPEN-\{n\}$，$CLOSED \leftarrow CLOSED \cup \{n\}$。

(4) 若 $n \in \{t_i\}$，算法以成功结束。

(5) 展开 n 并令 M 为 n 的子节点且不是其前辈节点的节点集。将 M 加入 G。

(6) 将 M 中既不在 *OPEN* 中也不在 *CLOSED* 中的节点放入 *OPEN*。

(7) 给每个 M 中既不在 *OPEN* 中也不在 *CLOSED* 中的节点设置一个指向其父节点的逆向指针。

(8) 调整 M 中那些同时也在 *OPEN* 或 *CLOSED* 中的节点的逆向指针。

(9) 调整 M 和 *CLOSED* 中每个节点的后继节点的逆向指针。

(10) 依据评价函数值重新安排 *OPEN* 中的节点。

(11) 回到 LOOP。

上述算法中所用的逆向指针就构成了解树 T，而回溯则是通过将节点分别放入 *OPEN* 和 *CLOSED* 两集合中而实现的。如果一个节点尚未被展开，亦即它的子节点还未被生成，则将它放入 *OPEN*；否则，一个已被展开的节点将被放入 *CLOSED*。图 2.4 示范出一个执行 Graph-Search 算法历经 3 次循环的例子。

循环 1

步骤(1)：最初 $OPEN=\{s\}$，*CLOSED* 为空集。见图 2.4(a)。

步骤(3)：s 被展开，*OPEN* 为空集，$CLOSED=\{s\}$。

步骤(5)：假定展开 s 生成 3 个子节点，$M=\{a,b,c\}$。见图 2.4(b)。

步骤(6)：$OPEN=\{a,b,c\}$。

步骤(7)：设置逆向指针。见图 2.4(c)。

循环 2

步骤(3)：假定 a 被选来扩展，$OPEN=\{b,c\}$，$CLOSED=\{s,a\}$。

步骤(5)：假定扩展 a 生成三个子节点，$M=\{b,d,e\}$，见图 2.4(d)。

步骤(6)：$OPEN=\{b,c,d,e\}$。

步骤(7)：设置逆向指针。见图 2.4(e)。注意 b 在此之前已有逆向指针指向 s，这种情况下不作调整。

循环 3

步骤(3)：假定 b 被选来扩展，$OPEN=\{c,d,e\}$，$CLOSED=\{s,a,b\}$。

步骤(5)：假定扩展 b 生成一个子节点，$M=\{e\}$。见图 2.4(f)。

步骤(6)：$OPEN=\{c,d,e\}$。

步骤(7)：设置逆向指针。见图 2.4(g)。注意 e 在此之前已有反向指针指向 a，这种情况下要调整为指向 b。