第3章


视频讲解

搜索的基本策略

本章主要讨论搜索的基本策略，即怎样搜索才可以最有效地达到目标。搜索的基本策略根据扩展的利用问题的特征信息的方式可分为盲目搜索、启发式搜索和随机搜索。如果没有利用问题的特征信息，一般的搜索方式与平时找东西在策略上可以说是相同的。

当我们在慌乱之中寻找东西的时候通常使用的就是随机搜索。

当我们在清醒时，有条理地寻找东西的方法大致可以分成两类：一种是找眼镜模式，它指的是眼镜掉了的时候总是从最近的地方开始寻找，慢慢地扩大搜索的范围；另一种是走迷宫模式，它指的是在走迷宫的时候由于无法分身只有一条路走到底，走不通再回溯的走法。

这3种方法分别对应的就是随机搜索、宽度搜索和深度搜索。

下面按是否利用问题的特征信息划分搜索策略的方法，讨论盲目搜索、启发式搜索和随机搜索。

3.1 盲目搜索

盲目搜索又叫非启发式搜索，是一种无信息搜索(uninformed search)，一般只适用于求解比较简单的问题。下面将要讨论的几个搜索方法，它们均属于盲目搜索算法，虽然其他课程也讨论类似的算法，但我们要探究人工智能的算法表达方法。

3.1.1 宽度优先搜索

在一个搜索树中，如果搜索是以同层邻近节点依次扩展节点的，那么这种搜索就叫宽度优先搜索(breath-first search)。这种搜索是逐层进行的，在对下一层的任一节点进行搜索之前，必须搜索完本层的所有节点。

在本节讨论的盲目搜索算法中存放节点都采用一种简单的数据结构——表，表是将节点按一定的顺序用逗号隔开放在一对括号中的一种数据结构，在表的首部和尾部都可以加入和删除节点。

宽度优先搜索算法流程如下。

(1) 令 N 为一个由初始状态构成的表。

(2) 若 N 为空退出，标志失败。

(3) 令 n 为 N 中第一个节点，将 n 从 N 中删除。

(4) 若 n 是目标，则退出，标志成功。

(5) 若 n 不是目标，将 n 的后继节点加入 N 表的尾部，转步骤(2)。

宽度优先搜索的优点：若问题有解，则可找出最优解。缺点：效率低，组合爆炸问题难以解决。

3.1.2 深度优先搜索

与宽度优先搜索对应的一种盲目搜索叫作深度优先搜索(depth-first search)。在深度优先搜索中，首先扩展最新产生的(即最深的)节点到表中。深度相等的节点可以任意排列。

深度优先搜索算法流程如下。

(1) 令 N 为一个由初始状态构成的表。

(2) 若 N 为空退出，标志失败。

(3) 令 n 为 N 中第一个节点，将 n 从 N 中删除。

(4) 若 n 是目标，则退出，标志成功。

(5) 若 n 不是目标，将 n 的后继节点加入到 N 表的首部，转步骤(2)。

深度优先搜索的优点：节省大量时间和空间。缺点：不一定能找到解。因为在深度无限搜索树的情况下，最坏的情况可能是不能停机。

宽度和深度优先搜索虽然在搜索的策略上走了两个极端，但是它们在控制策略上的差异并不大。它们大都假设以队列作为数据结构，每次选队列的第一个节点进行扩展。宽度和深度优先搜索的区别在于：宽度优先搜索把结果存在队列的尾部；而深度优先搜索则是把它存在首部，只有一字之差。

3.1.3 分支有界搜索

分支有界搜索(branch-and-bound search)也是一种深度优先搜索，但每个分支都规定了一个统一的搜索深度，搜索到这个深度后，如果没有找到目标便自动退回到上一层，继续按深度优先搜索。其算法流程如下。

(1) 令 N 为一由初始状态构成的表。

(2) 若 N 为空退出，标志失败。

(3) 令 n 为 N 中第一个节点，将 n 从 N 中删除。

(4) 若 n 是目标，则退出，标志成功。

(5) 若 n 深度为预先定好的一个界 $d_{\max}$，则转步骤(2)。

(6) 若 n 不是目标，将 n 的后继节点加入 N 表的首部，转步骤(2)。

此方法若被搜索树的深度远大于目标点的深度，则快于深度优先搜索。

3.1.4 迭代加深搜索

迭代加深搜索(iterative deepening search)是在分支有界搜索的基础上，对 $d_{\max}$ 进行迭代，即逐步加深。这是一种同时兼顾深度和宽度的搜索算法。在限定的深度内，保证了对宽度

节点的搜索，如果没有找到解，则再加深深度。

3.1.5　一个盲目搜索问题的几种实现

这里给出一个简单的盲目搜索问题：对于中国象棋，如果“马”（棋子的名称）当前所在位置是 cp(x,y)，它跳一步可能到达的位置最多有 8 个，如图 3-1 所示。

要求设计一个算法，对于任意给定的棋盘上的坐标位置 tp，输出马从当前位置 cp 出发通过搜索到达的该坐标位置 tp。

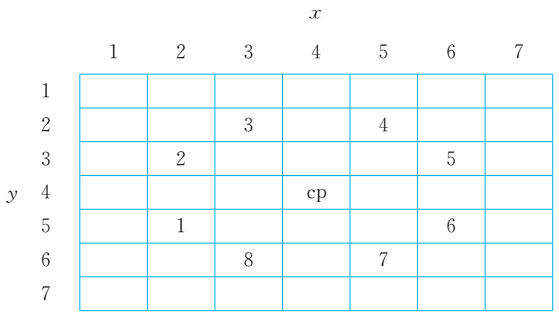


图 3-1　中国象棋中跳马可能到达的 8 个位置

求解过程如下。

1. 定义状态空间

设状态空间的一点为 10×9 矩阵。

2. 定义操作规则

马走棋盘可以模拟为一个搜索过程：每到一处，总让它按东、东南、南、西南、西、西北、北、东北 8 个方向顺序试探下一个位置；如果某方向可以通过，并且在前面的搜索过程中不曾到达，则可以前进一步；在新位置上按上述方法就可以重新进行搜索，并将到达的位置标记 * 。

对这 8 个方向，从西开始，按顺时针的顺序依次编号为 1,2,…,8。算法中，设计 move 数组是一个二维数组，记录了 8 个方向上的行下标增量和列下标增量。如果当前所在位置是(x,y)，则沿第 i($0 \leqslant i \leqslant 7$)个方向前进一步，到达新位置($x_1$,$y_1$)，其下标可以借助 move 数组确定：

x_1 = x + move[i][0]
y_1 = y + move[i][1]

3. 定义搜索策略

为了避免重复，可以对搜索过的地方做一个标记，不同的搜索分支可以避开搜索这个地方。

(1) 宽度优先搜索。搜索过程为：沿着 8 个方向，如果可行，都前进一步，看是否达到位置 tp。如果没有达到，则依次从新的位置为起点，沿着 8 个方向继续前进一步……直到搜索到目标位置 tp，或找不到未搜索的位置为止。

(2) 深度优先搜索。搜索过程为：沿着 8 个方向中的某一个。例如，从朝东方向开始，前进一步，看是否达到位置 tp。如果没有达到，则以这个位置为起点，沿着 8 个方向中的某一个继续前进一步……某个分支搜索不通时，再沿着当前位置 8 个方向的下一个方向继续搜索。直到搜索到目标位置 tp，或找不到未搜索的位置为止。

(3) 分支有界搜索。搜索过程为：沿着 8 个方向中的某一个，例如，从朝东方向开始，前进

一步，看是否达到位置 tp。如果没有达到，则以这个位置为起点，沿着 8 个方向中的某一个继续前进一步……如果搜索到第 k 步或某个分支搜索不通时，再沿着当前位置 8 个方向的下一个方向继续搜索。直到搜索到目标位置 tp，或所有的分支都搜索到了第 k 步，或找不到未搜索的位置为止。

3.2 启发式搜索

盲目搜索的效率低，耗费过多的搜索时间。如果能够找到一种方法选择最有希望的节点加以扩展，那么，搜索效率将会大大提高。启发式搜索就是基于这种想法，它是深度优先搜索的改进。搜索时不是任取一个分支，而是根据一些启发式信息，选择最佳的一个或几个分支往下搜索。

3.2.1 启发式信息的表示

1. 启发式搜索的依据

启发式搜索作为一种基本的搜索方法，其主要根据如下。

（1）人们善于利用一些线索来帮助自己选择搜索方向，这些线索统称为启发式（heuristics）信息，heuristics 一词来源于希腊语 heuriskein，即“发现”的意思。

（2）现实问题往往只需一个解，而不要求最优解或全部解。

（3）使用启发式信息可以避免某些领域里的组合爆炸问题。

例 3.1 计算机下棋问题。若采用将所有可能的走法都试一遍，并从中找出最佳的一步，下面几种棋的可能走法分别为：

一字棋：9!步。

国际象棋：10^{120} 步，如果计算机以最快速度一年走 10^{104} 步，需 10^{16} 年才能完成。

围棋：10^{761} 步，需要的时间更长。

像这样一类问题必须利用启发式信息来求解。这些问题的初始状态、算子和目标状态的定义都是完全确定的，并且有一个确定的搜索空间。现在的问题就在于如何有效地搜索这个空间，因为这个空间太大。进行这种搜索的技术一般需要某些有关具体问题领域的特性的信息。这些信息常常可以用来简化搜索。把这种信息叫作启发式信息，并把利用启发式信息进行的搜索方法叫作启发式搜索方法。启发式信息按其形式可分为下列两种。

（1）表示为估计函数。确定一个启发式函数 $f(n)$，n 为被搜索的节点，$f(n)$把 n 所处的问题状态的描述映射成问题解决的程度，通常这种程度用数值来表示，就是启发式函数的值。这个值的大小用来决定最佳搜索路径。

（2）表示成规则。如 Lennat 编制了一个程序 AM，从数论、集合论的一些基本概念出发，发现了标准数论大量的定理，其中一条启发式规则如下：

如果存在一个有趣的二元函数 $f(x,y)$，那么看看两变元相同时会发生什么？

若 f 表示乘法：导致发现平方。

若 f 表示集合并运算：导致发现恒等函数。

若 f 表示思考：导致发现反省。

（3）表示成元规则。还有一些启发式规则用于控制使用状态空间中的搜索规则（或算子），即这个规则体现了状态空间的搜索策略，这个策略用到启发式信息。这样的启发式规则

也称为元规则(meta rule),是如何使用规则的规则。

2. 启发式函数的表示方法

启发式函数是一种映射函数,它把对问题的当前状态的描述映射成一种接近目标的程度。使用这个函数,可以引导人们解决当前的问题。在搜索算法的设计中,考虑问题的状态的各种因素、各种特征,都可用于启发式函数的设计,只要这些因素能够帮助该函数对节点在解路径上作出准确的估计。启发式函数设计得好,对有效引导搜索过程有着重要的作用。在有的情况下,非常简单的启发式函数搜索路径能够作出非常令人满意的估计,对有些场合还需要使用非常复杂的启发式函数。下面就简单讨论一下如何构造启发式函数。

(1) 设计的启发式函数应能够根据问题的当前状态,确定用于继续求解问题的信息。

例 3.2　考虑利用启发式信息去求解水壶问题(即如何通过两个容量为 4 升、3 升的水壶得到 2 升水),人们绝不盲目搜索,而是利用水壶容量信息 4,3,0,考虑如何求得 2。用这几个数字可以考虑 4 减 2 或 3 减 1 得到 2。但考虑 4 减 2 肯定不行,因为它用到 2 来求得 2,用后一个方法要考虑如何求得 1。这很容易在当前信息中找出,因为 4 减 3 等于 1。由于这里的"减去"是从某个水壶中向外倒水,可以从加满水的 4 升水壶里向空的 3 升水壶里加满,这样 4 升水壶里只剩下 1 升。这样得到数字 1。再从现有数字中不难想到 3－1＝2。由 3－1＝2 又可以想到把 3 升水壶倒空,再把 4 升水壶的 1 升水倒入 3 升水壶中,3 升水壶还缺 2 升水,现在有数字 2 了,可以考虑 4－2＝2。也就是将 4 升水壶的水加满,然后用 4 升水壶中的水将 3 升的水壶加满。这样 4 升水壶中就只剩下 2 升了。用只有两个元素的表:(4 升水壶的水量,3 升水壶的水量),上述的求解路径可表示为

$$(0,0)\rightarrow(4,0)\rightarrow(1,3)\rightarrow(1,0)\rightarrow(0,1)\rightarrow(4,1)\rightarrow(2,3)=\text{目标}$$

其搜索过程如图 3-2 所示。

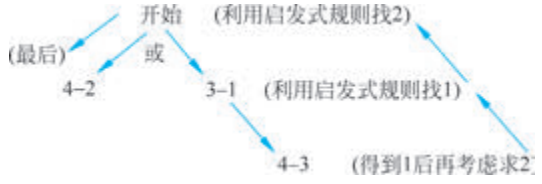


图 3-2　利用启发式信息求解水壶问题的搜索过程

(2) 设计的启发式函数应能够估计已找到的状态与达到目标的有利程度。这样的启发式函数能够有效地帮助决定哪些后继节点应被作为下一步搜索的起点。

例 3.3　8 数码问题。

初始状态 S_0 $\begin{bmatrix}2&8&3\\1&6&4\\7&&5\end{bmatrix}$　　目标状态 S_g $\begin{bmatrix}1&2&3\\8&&4\\7&6&5\end{bmatrix}$

一般状态,即问题空间为 $\begin{bmatrix}a_{11}&a_{12}&a_{13}\\a_{21}&a_{22}&a_{23}\\a_{31}&a_{32}&a_{33}\end{bmatrix}$

各状态间的转换规则为:

Pr1：空格上移 ↑

If $\square_{i,j}$ and $i \neq 1$ then $a_{i-1,j} \longleftrightarrow \square_{i,j}$

Pr2：空格下移 ↓

If $\square_{i,j}$ and $i \neq 3$ then $a_{i+1,j} \longleftrightarrow \square_{i,j}$

Pr3：空格左移 ←

If $\square_{i,j}$ and $j \neq 1$ then $a_{i,j-1} \longleftrightarrow \square_{i,j}$

Pr4：空格右移 →

If $\square_{i,j}$ and $j \neq 3$ then $a_{i,j+1} \longleftrightarrow \square_{i,j}$

其中，Pr 表示产生式规则（production rule），它类似于有限自动机的转换规则；$a_{i+1,j} \longleftrightarrow \square_{i,j}$ 表示处在第 $i+1$ 行第 j 列的元素与第 i 行第 j 列的空格交换。

定义启发式函数 f_1＝数字错放位置的个数。$f_1=0$，则达到目标。其搜索过程如图 3-3 所示。

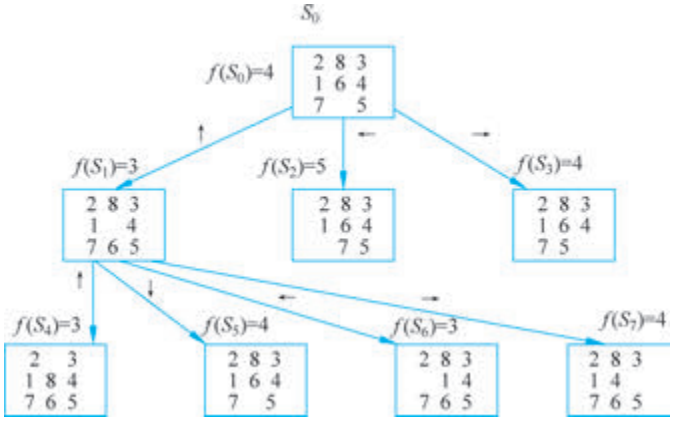


图 3-3 利用启发式函数求解 8 数码问题的搜索过程

但利用这个启发式函数搜索到第 3 层时，S_5 是应避开的（不应生成的），又由于 $f_2(S_4)=f_2(S_6)=3$，这称为均势（tie），因此下一步还得确定一个策略，以决定当 f_1 值相同时如何打破均势（break tie），向下搜索。这意味着要定义一个新的启发式函数。由此可以看出，同一问题启发式函数可能有多种选择。

现在定义新的启发式函数为

f_2＝所有数字当前位置次短路径走到正确位置的步数之和

在这个定义之下，各状态的启发式函数值为

数码 1 2 3 4 5 6 7 8

$f_2(S_0)=1+1+0+0+0+1+0+2=5$

$f_2(S_1)=1+1+0+0+0+0+0+2=4$

$f_2(S_2)=1+1+0+0+0+1+1+2=6$

$f_2(S_3)=1+1+0+0+1+1+0+2=6$

$f_2(S_4)=1+1+0+0+0+0+0+1=3$

$f_2(S_5)=1+1+0+0+0+1+0+2=5$

$f_2(S_6)=1+2+0+0+0+0+0+2=5$

其中，每个函数中加法的每一项表示对应数码移动的步数。

现在可以看出，原来有 $f_2(S_4)=f_2(S_6)=3$，现在 $f_1(S_4)<f_1(S_6)$，所以下一步搜索从 S_4 出发，产生其后继节点。

(3) 设计的启发式函数应能够估计出可能加速达到目标的程度，它最直接的作用是当扩展一个节点时，能帮助确定哪些节点应从搜索树中删除。

启发式函数对搜索树(图)中每一节点的真正优点估计得越精确，解题过程就越少走弯路。

例 3.4　8 皇后问题(8-Queens Problem)。

在 8×8 棋盘中放下 8 个皇后，要求没有一个皇后能够攻击其他皇后，即要使得在任何一行、一列或对角线上都不存在两个或两个以上的皇后。

求解这个问题的过程如下：

(1) 定义状态空间。设状态空间的一点为 8×8 矩阵。

(2) 定义操作规则。为简单起见，可按以下规则放置皇后：

第一个皇后放第一行。

第二个皇后放在第二行且不与第一个皇后在同一列或对角线的空格上。

……

第 i 个皇后放在第 i 行且不与前面 $i-1$ 个皇后在同一列或对角线的空格上。

……

按照这个规则，在某一步放置某个皇后时可能有若干空格可以使用，究竟放哪一个空格，可使用以下启发式函数：

$$f(x)=\text{剩下未放行中能够用来放 } Q(\text{Queen})\text{的空格数}$$

其中，x 为当前要放置 Q 的那个空格。

例如，在放置了 3 个 Q 后，第 4 个 Q 可放在第 4 行的 A、B、C 3 个位置。

在图 3-4 中，记：a 为第 4 行 A 处放了皇后，剩下可放 Q 的位置；b 为第 4 行 B 处放了皇后，剩下可放 Q 的位置；c 为第 4 行 C 处放了皇后，剩下可放 Q 的位置。

			Q				
	Q						
							Q
		A		B	C		
abc		bc				ab	
bc		c				ac	
abc		b		ac	b		
ac				ac	ab	bc	

图 3-4　8 皇后问题放置了 3 个 Q 后能够继续放 Q 的位置

按照以上定义，可求得：$f(A)=8$，$f(B)=9$，$f(C)=10$。不难看出，$f(x)$ 越大越好，应选择 $f(x)$ 最大的空格来放置皇后。所以，搜索可以从 C 对应的空格放置一个皇后开始，其余的空格对应的搜索树可以删除。

(3) 定义搜索策略。

第 i 个皇后放到第 i 行中的那个与前面 $i-1$ 个皇后不在同一列或对角线上且 $f(x)$ 值最大的空格中。

由此可见，启发式信息是某些领域里的知识信息，它能使计算机系统在这些知识信息提示以后可能采取的某些可能的动作或避免某些不可能的动作，它能够说明动作适合的条件或不适合的条件。所以，人们把通过启发式信息引导的搜索方法称为启发式搜索方法，并定义为一种专门的技术，以帮助问题求解。

启发式搜索方法按其适用的范围也可以分为两种：一种是适用于特定的领域，如果这个领域很窄，这种启发式搜索方法就变成了类似于计算方法中的算法；另一种是适用于较广泛领域的通用启发式搜索算法，这类算法是后面要讨论的弱法。

3. 搜索方向的选择

搜索过程的目的在于：在问题空间中找出从开始状态到目标状态的一条最好的或较好的路径。这种搜索可按两个方向进行：正向搜索，从初始状态朝着目标状态方向搜索；逆向搜索，从目标状态朝着初始状态方向搜索。

将以上两种搜索方法结合起来，就产生了双向搜索，即同时从初态和目标态出发，朝着对应的方向搜索。

正向搜索和逆向搜索可用图 3-5 来说明。



(a) 正向搜索　　(b) 逆向搜索

图 3-5　正向搜索和逆向搜索

一般来说，采用双向搜索是为了提高搜索效率。理想的双向搜索如图 3-6(a)所示，而使用不当的双向搜索可能产生图 3-6(b)所示的结果。这种结果不会比单独使用正向搜索或逆向搜索好，因为它总是取正向搜索和逆向搜索中最差的结果。



(a) 理想的双向搜索的结果　　(b) 最差的双向搜索的结果

图 3-6　双向搜索结果

究竟采用正向搜索好还是采用逆向搜索好，可以考虑以下因素。

(1) 朝分支因子低的方向更有效。

分支因子指从一点出发可以直接到达的平均节点数。朝着分支因子低的方向搜索意味着朝着“收敛”的方向搜索。例如，定理证明，一般是从公理或定理出发，推出新的定理。公理是有限的，而定理是大量的。如果采用正向搜索，就要从大量可行的定理中寻找待证的定理，如图 3-7(a)所示。很多定理并不是需要的，从公理或定理出发可以证明出大量的定理，但意义不大。要找出感兴趣的定理，也许是大海捞针。所以，要从被证明的定理出发，即逆向搜索，找出支持这个定理的公理和定理。这就是朝着分支因子低的方向搜索，如图 3-7(b)所示。

(2) 由状态少的一方出发，朝着大量的可识别的状态的方向搜索，会容易一些。例如，符号积分问题，正向搜索意味着从被积函数出发，按照积分规则，寻找原函数，如图 3-8(a)所示。而逆向搜索，则要从大量的原函数的任意组合出发，通过积分规则，找出被积函数，如图 3-8(b)所示。这显然要困难得多，在人工演算积分问题时绝不会这么去做。分析一下这两

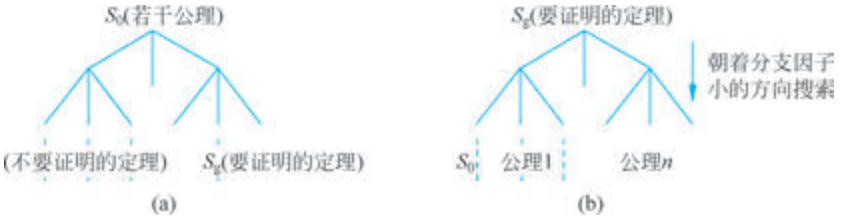


图 3-7　定理证明的两个方向搜索

种情况，已知的被积函数，它的状态数是很少的，而原函数的任意组合的数目则是大量的，但是每个状态是可识别的(原函数可以通过查表得到)，该问题采用正向搜索，就是由状态少的一方出发，朝着大量的可识别的状态的方向搜索。

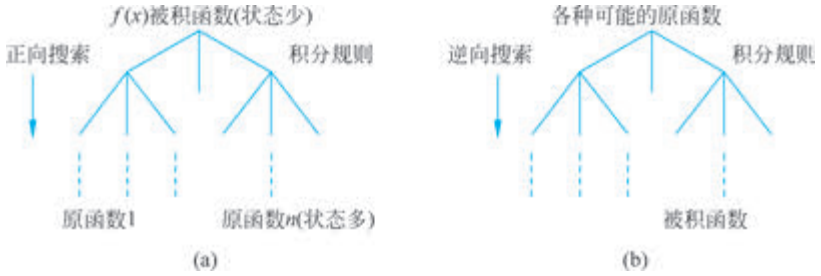


图 3-8　积分问题的两个方向搜索

(3) 依据用户可接受的方向。特别是需要向用户解释推理过程时，顺应用户的心理，选择搜索方向会使系统显得更自然一些。在建造专家系统时，向用户解释为什么系统会得出某个结论，这一步骤是必不可少的。

3.2.2　几种最基本的搜索策略

下面主要介绍几种基本搜索策略，这些策略已构成了许多人工智能系统的构架，其搜索效率取决于问题所在领域知识的利用与开发。由于这些方法的通用性，并且难以克服搜索过程的组合爆炸问题，所以又称为弱(weak)法。下面讨论几种使用弱法的通用问题求解策略。

1. 生成测试法

生成测试法(generate-and-test)的基本步骤如下。

(1) 生成一个可能的解，此解是状态空间一个点，或一条始于 S_0 的路径。

(2) 用生成的“解”与目标比较。

(3) 达到目标则停止；否则转步骤(1)。

此方法属于深度优先搜索，因为要产生一个完全的解后再判断。若不是目标，则要生成下一个“解”。这种方法几乎接近耗尽式搜索，因而效率低。于是，人们考虑能否利用反馈信息以帮助决定生成什么样的解，这种改进就是下面要讨论的爬山法。

2. 爬山法

爬山法(hill-climbing)又可比喻为盲人爬山法，之所以这样称谓是因为这种方法只用到局部的反馈信息，就像盲人只能用他的拐棍来探测周围(局部)的高低情况，试探着爬山。爬山法的基本步骤如下。

（1）生成第一个可能的解。若是目标则停止；否则转下一步。

（2）从当前可能的解出发，生成新的可能解集。

① 用测试函数测试新的可能解集中的元素，若是解，则停止；若不是解，转步骤②。

② 将它与至今已测试过的“解”相比较。若它最接近解，则保留作为最佳元素；若它不最接近解则舍弃。

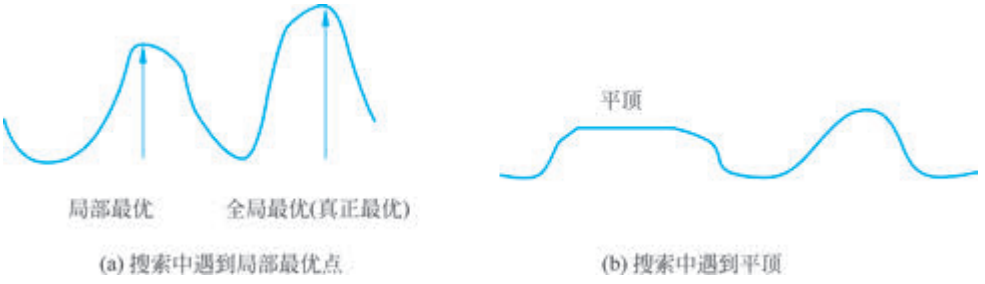
（3）以当前最佳元素为起点，转步骤(2)。

爬山法在生成的元素中寻找最优解，这种最优是局部最优。爬山法会产生下述问题：

（1）找到的是局部最大值，如图 3-9(a)所示。

（2）碰到平顶时无法处理，如图 3-9(b)所示。

遇到平顶是在某一局部点周围 $f(x)=\text{constant}$，此时无法决定要搜索的最佳方向。



(a) 搜索中遇到局部最优点　　(b) 搜索中遇到平顶

图 3-9　爬山法中出现的问题

（3）碰到山脊时无法处理。山脊是在搜索空间中的一块“带状地区”，在此带状地区的 $f(x)$值与周围邻域内的值相等。

碰到山脊的克服办法是：退回较大一步，即允许回溯；向前跨一大步；多设几个初始点，从几个初始点同时或先后进行搜索。

3. 最佳优先搜索法

最佳优先搜索(best-first search)法是由爬山法改造而来的，其算法步骤如下。

（1）生成第一个可能的解。若是目标则停止；否则转步骤(2)。

（2）从该可能的解出发，生成新的可能解集。

① 用测试函数测试新的可能解集中的元素，若是解，则停止；若不是解，转步骤②。

② 将新生成的“解”集加入原可能“解”集中。

（3）从解集中挑选最好的元素作为起点，再转步骤(2)。

最佳优先搜索是从“所有可能解集”中找最佳值，因此是全局最佳。爬山法是从当前后继中求最好的，因此，比爬山法考虑的因素要多些。

爬山法与最佳优先搜索法的区别如图 3-10 所示，其中，$f(x)$为 x 到目标距离的估计函数。

在上述问题中，若解在 G 处，则 $f(G)=0$，爬山法没有办法找到解；若解在 I 处，则 $f(I)=0$，此时爬山法可比最佳优先搜索法更快地找到解，因为对最佳优先搜索法来说，必须从 B 分支开始搜索，由于 G、H 都不是解，因而 f 值必然较大，最后还是必须转移到 $D\rightarrow E\rightarrow I$ 的分支上来。

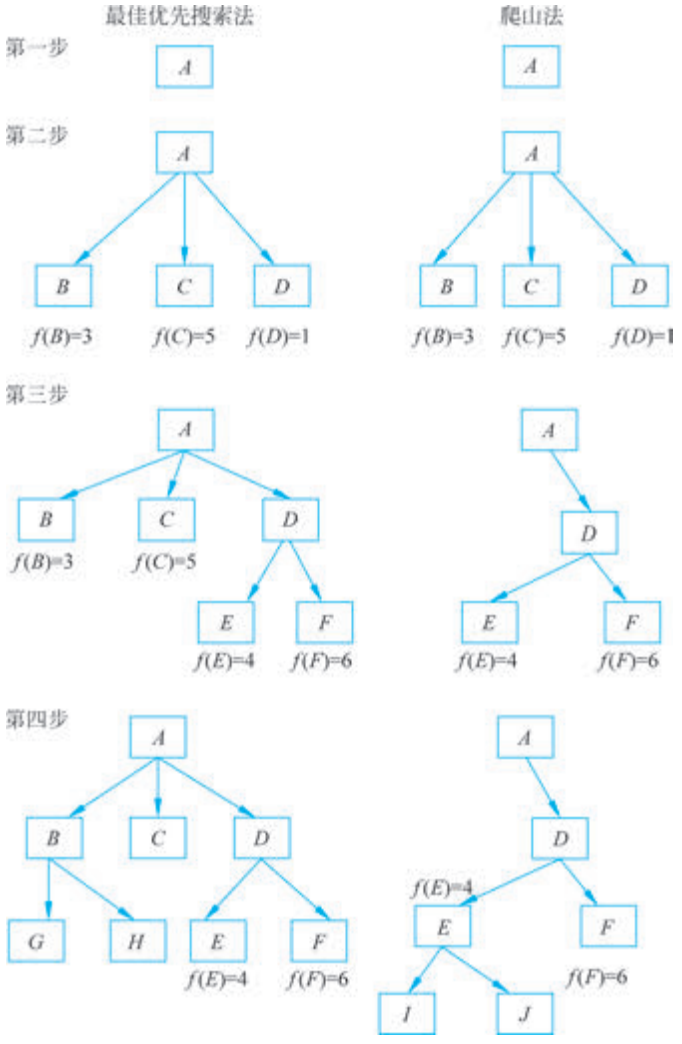


图 3-10　最佳优先搜索法与爬山法的比较

3.3　随机搜索

虽然随机搜索看上去效率很低，但是其实当问题的空间很大，而可行解较多，并且对解的精度要求不高时，随机搜索是很有效的解决办法，因为其他的做法在这个时候的时空效率不能让人满意。而借助演化思想和群集智能思想改进过的随机算法更是对解的分布呈现一定规律性的复杂问题有良好的效果。

3.3.1　模拟退火法

模拟退火法(simulated annealing)是克服爬山法缺点的有效方法。所谓退火，是指冶金专家为了达到某些特种晶体结构重复将金属加热或冷却的过程，该过程的控制参数为温度 T。模拟退火法的基本思想是，在系统朝着能量减小的趋势这样一个变化过程中，偶尔允许系统跳到能量较高的状态，以避开局部极小点，最终稳定到全局最小点。如图 3-11 所示，若使能量在 C 点突

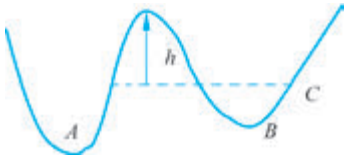


图 3-11　增加能量 h 跳过局部极小点 B 找到全局最小点 A

然增加 h，就能跳过局部极小点 B，而找到全局最小点 A。

现在的问题是何时增加能量？应该增加多少能量？为此，柯克帕特里克(S. Kirkpatrick)提出了模拟退火法。

模拟退火法步骤如下。

(1) 随机挑选一单元 k，并给它一个随机的位移，求出系统因此而产生的能量变化 ΔE_k。

(2) 若 $\Delta E_k \leqslant 0$，则该位移可采纳，而变化后的系统状态可作为下次变化的起点；若 $\Delta E_k > 0$，则位移后的状态可采纳的概率为

$$P_k = 1/(1+\mathrm{e}^{-\Delta E_k/T})$$

式中，T 为温度，然后从(0,1)区间均匀分布的随机数中挑选一个数 R。若 $R < P_k$，则将变化后的状态作为下次的起点；否则，将变化前的状态作为下次的起点。

(3) 转步骤(1)继续执行，直至达到平衡状态为止。

概率分布稳定并不意味着系统达到的仅仅是单一稳定状态。不过，对一具体温度 T 而言，达到平衡时任两个状态 α 与 β 的概率 P_α 与 P_β 均服从玻尔兹曼分布，则

$$\frac{P_\alpha}{P_\beta} = \mathrm{e}^{-(E_\alpha - E_\beta)/T}$$

式中，E_α、E_β 为 α、β 两个状态的能量。

公式表明：若 $E_\beta > E_\alpha$，则 $\dfrac{P_\alpha}{P_\beta} > 1$，说明能量越小，平衡状态概率越大，因而系统处于能量较小的平衡状态的可能性也越大。另外，温度 T 对系统的影响如下。

① 温度越高，系统越容易达到平衡状态，但却使 P_α/P_β 值越小，故相对而言，在平衡状态下处于能量较小状态的可能性也越小。

② 温度越低，系统达到平衡状态的速度虽慢，但 P_α/P_β 值越大，故系统可能达到能量较小的平衡状态。

在搜索问题中，引入模拟退火法的爬山法，不仅可以避免陷入局部最优，而且能够通过概率机制在搜索过程中允许更大步长的随机变化，从而增强算法的全局搜索能力。通常的做法是：最初阶段倾向于取大步；后续阶段倾向于取小步。



图 3-12　使小球跳过局部极小点

这个过程的原理可以根据图 3-12 所示形象地去理解，如果希望小球离开 A 点然后停在 B 点(全局最小)，使用较小的能量来摇动系统，这小球只能停在 A 点。若开始以较大的速度摇动，后来慢慢地减轻，则小球很可能就会落在 B 点，且小球到 B 点之后，就不易从 B 点摇到 A 点。模拟退火法类似于这个过程。

3.3.2　其他典型的随机搜索算法

在许多搜索算法中，采取的搜索策略并非只是一种，而随机搜索就是采取的策略之一。采用随机搜索的目的往往是增加算法的灵活性和搜索过程扩展方式的多样性，使得算法避免陷入过早收敛的境地。这样的一些算法常常给启发式函数加入一些带有随机性的调控参数，例如参数中有 rand()函数或其他随机控制手段，这类算法有遗传算法、粒子群算法和蚁群算法等。

1. 遗传算法

遗传算法(genetic algorithm)的基本思想来源于达尔文的进化论。达尔文认为：每个物

种出生个体的数目总是比能够生存下来的个体数目多，因此个体之间为了生存而相互竞争。如果某个个体的特征发生微小的变异，尽管很小，但使得其适应能力有所提高，那么在复杂的、不断变化的自然环境中，这个个体就有更大机会生存下来，这就是自然选择(natural selection)。变异得到的特征经过遗传由后代继承。通过遗传、变异和自然选择，生物物种能够不断进化。

遗传算法则是模拟生物进化的自然选择和遗传机制的一种随机搜索方法，适用于复杂的非线性问题。该算法的主要步骤为：编码；产生初始种群；计算适应度；选择；交叉；变异。

由于遗传算法只使用目标函数(适应值)进行搜索，可以处理很多类型的问题。遗传算法使用的遗传算子是一种随机操作，而不是确定性规则，其中选择、交叉和变异操作是由一定概率来控制的。

2. 人工免疫算法

人工免疫算法是模拟了人体的免疫细胞的工作机制设计出来的算法。它和遗传算法相比，只是把遗传算子和控制参数的部分操作做了一些改变。

该算法从种群中选择适应值最高的一批个体并对它们进行变异操作，而这里变异操作的概率随着适应度的增加而减少。在原来的种群中把一部分适应度差的个体淘汰掉，并从变异完成的个体中找出适应度最高的那部分组成新的种群。

该算法的框架也与遗传算法基本相似，但有的只采用了变异操作和选择操作，有的又增加了一些操作。它增加了群体的多样性，保留了更多且不同的最优个体，随进化过程的进行而不断更新这些个体，这样能加速算法找到全局最优值。经仿真实验证明在优化几个相关的概率后，对很多问题该算法的处理效率比遗传算法要好。

3. 蚁群算法

蚁群算法于20世纪90年代早期由马可·多里戈(Marco Dorigo)等提出发展并完善。蚁群算法模拟蚂蚁从巢穴出发，通过对信息素(pheromone)的追踪来找到从巢穴到食物之间的最短路径。模拟过程中假定：

① 每只蚂蚁都是随机移动的。

② 信息素被洒到经历过的路径上。

③ 蚂蚁能感知周围的信息素。

④ 一条路径上的信息素的浓度越高，则其被其他蚂蚁选择的可能性越大。

模拟过程由以下6个规则控制。

(1) 范围。蚂蚁观察到的范围是一个方格世界，蚂蚁有一个参数为速度半径(一般是3)，那么它能观察到的范围就是3×3个方格世界。

(2) 环境。蚂蚁所在的环境是一个虚拟的世界，其中有障碍物，有别的蚂蚁，还有信息素。信息素有两种，一种是找到食物的蚂蚁撒播的食物信息素，另一种是找到窝的蚂蚁撒播的窝的信息素。环境以一定的速率让信息素消失。

(3) 觅食规则。在感知范围内寻找是否有食物，如果有就直接过去；否则看是否有食物信息素，并且比较在能感知的范围内哪一点的信息素最多，就朝信息素多的地方走。由于每只蚂蚁都有小概率犯错误的可能，所以并不是总往信息素最多的点移动。

(4) 移动规则。每只蚂蚁都朝向信息素最多的方向移动，并且当周围没有信息素指引时，蚂蚁会按照自己原来运动的方向惯性地运动下去，并且在运动的方向上有一个随机的小扰动。

(5) 避障规则。如果蚂蚁要移动的方向有障碍物挡住，它会随机地选择另一个方向，并且有信息素指引的话，它会按照觅食/找窝的规则行动。

（6）播撒信息素规则。在不同的蚁群优化算法中，有的蚂蚁每次撒播的信息素是常量，有的蚂蚁撒播的信息素是变量，但是这些信息素都随时间动态变化、逐渐消失。

从模拟过程中的假设和控制规则可以看出，随机搜索是该搜索算法的主要特点之一。

4. 粒子群算法

粒子群算法简称 PSO（Particle Swarm Optimization），它具有蚁群算法和遗传算法两者的特点，它和蚁群算法一样采用的是增量方式进行搜索，但是在结构上，不论是种群的初始化、适应性函数、终止条件，还是其他方面和遗传算法是基本一致的。

PSO 中“粒子”就相当于遗传中的个体，和个体不同的是每个粒子有一个速度决定它们飞翔的方向和距离，然后粒子们就追随当前的估价函数评出的最优粒子在解空间中搜索。

PSO 初始化和遗传算法一样，然后通过迭代找到最优解。在每一次迭代中，粒子通过跟踪两个“极值”来更新自己。第一个极值就是粒子本身所找到的最优解，这个解叫作个体极值 pbest；另一个极值是整个种群目前找到的最优解，这个极值是全局极值 gbest。在找到这两个最优值时，粒子根据以下公式来更新自己的速度和新的位置，即

$$v[]=w*v[]+c_1*\text{rand}()*(\text{pbest}[]-\text{present}[])+c_2*\text{rand}()*(\text{gbest}[]-\text{present}[]) \tag{3-1}$$

$$\text{present}[]=\text{present}[]+v[] \tag{3-2}$$

其中，$v[]$是粒子的速度；present[]是当前粒子的位置；rand()是介于(0,1)之间的随机数；c_1、c_2 是学习因子；w 称为惯性因子，w 较大适于对解空间进行大范围探查，w 较小适于进行小范围开挖，w 越小收敛得越快，但是容易收敛于局部极值，所以通常采取随着迭代的加深线性减少 w 的取值的方法来进行优化。

这种算法有实现容易、精度高、收敛快的特点；而它的弊端和遗传算法一样，容易收敛到局部的极值。

5. 混合随机算法

一般弥补随机算法缺陷的方法有两种：一种是提高自身算法，比如调整自身算法控制机制、优化自身算法参数等；另一种就是混合其他算法的优势来提升自己的性能。以下是几种常见的混合方式。

（1）模拟退火与遗传算法及 PSO 的结合。PSO 和遗传算法都有十分强的全局搜索能力，但是容易陷入局部最优值，而模拟退火的 metropolis 法则可以帮助它们跳出局部最优，得到更快的收敛速度。

（2）蚁群算法与遗传算法及 PSO 的结合。它们的互补在于局部搜索和全局搜索。结合分两种：一种以蚁群算法为基础；另一种以遗传算法为基础。第一种思想主要是在蚁群中加入交叉和变异算子来实现，或是由遗传算法计算得出大致的最优点分布，将其作为蚁群的信息素出现。第二种策略是先用遗传算法计算得出一个可行解，再用蚁群算法来进行局部优化或是让蚁群算法改善变异算子。

以上这些算法还将用单独的章节分别予以详细讨论。

习题 3

3.1　对 $N=5, k\leqslant 3$ 的传教士和野人问题，定义两个 h 函数（非零），并给出用这两个启发式函数的 A 算法搜索图。讨论用这两个启发式函数求解该问题时是否能得到最佳解。

3.2　在3×3的空格内，用1,2,…,9的9个数字填入9个空格内，使得每行数字组成的十进制数平方根为整数。试用启发式搜索算法求解，分析问题空间的规模和有用的启发式信息。

3.3　试给出爬山法和最佳优先搜索算法搜索图3-13所示的搜索路径。

3.4　找出深度优先搜索算法与广度优先搜索算法在文字叙述中的区别。

3.5　在哪种问题空间，深度优先搜索好于宽度优先搜索？

3.6　在什么时候最佳优先搜索比宽度优先搜索差？

3.7　考虑将图3-14(a)中的积木形式转成图3-14(b)中的积木形状这一问题，可以使用的操作符有PICKUP、PUTDOWM、STACK和UNSTACK。用爬山法求解这一问题有解吗？为什么？

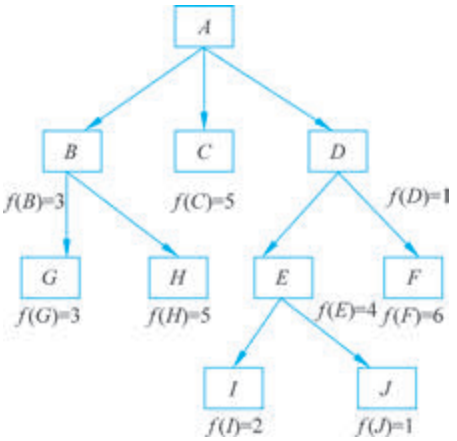


图3-13　搜索路径

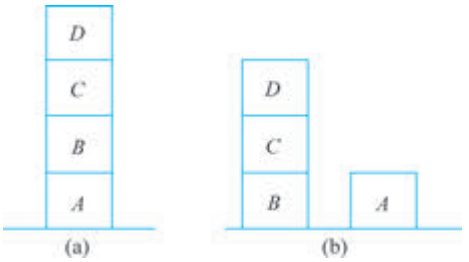


图3-14　习题3.7用图

3.8　一个经理有3个女儿，3个女儿的年龄加起来等于13，3个女儿的年龄乘起来等于经理自己的年龄，有一个下属已知道经理的年龄，但仍不能确定经理3个女儿的年龄，这时经理说只有一个女儿在上学，然后这个下属就知道了经理3个女儿的年龄。请问3个女儿的年龄分别是多少？为什么？请找出启发式信息。

3.9　假设要写一道问题求解搜索程序去求解下面的每一类问题，试确定搜索是正向进行还是逆向进行：a.模式识别；b.积木世界；c.语言理解。

3.10　对下面的每类问题，试描述一个好的启发式函数：a.积木世界；b.定理证明；c.传教士和野人。

3.11　分析宽度优先搜索和深度优先搜索的优、缺点，举出它们的正例和反例。

3.12　有一个农夫带一只狐狸、一只小羊和一篮菜过河。假设农夫每次只能带一样东西过河，考虑安全，无农夫看管时，狐狸和小羊不能在一起，小羊和菜篮不能在一起。试设计求解该问题的状态空间，并画出状态空间图。



习题3