第 3 章思维导图

思政引领

人工智能是新一轮科技革命的核心，是信息技术的集大成者，它不仅是一种工具，更是一种不断迭代更新的环境。人工智能问题，广义地说，都可以看作一个问题求解过程，它通常是通过在某个可能的解答空间中寻找一个解来进行的。本章通过对智能搜索策略的介绍，培养学生与时俱进的探索精神和爱国主义情怀。

第3章　智能搜索策略

人工智能问题广义地说，都可以看成一个问题求解过程，它通常是通过在某个可能的解空间中寻找一个解来进行的。在问题求解过程中，人们所面临的大多数现实问题往往没有确切性的算法，通常需要用搜索算法来解决。本章将主要介绍问题求解过程的形式表示和几种常用的搜索策略。


微课视频

3.1　搜索概述

人工智能的研究对象经常会涉及用常规算法无法解决的两类问题：一类是结构不良或非结构化问题；另一类是结构较好，理论上也有算法可依，但问题本身的复杂性超过了计算机在实践、空间上的局限性的问题。对于这些问题，一般很难获得其全部信息，更没有现成的算法可供求解使用，因此只能依靠经验，利用已有知识逐步探索求解。将这种根据实际情况，不断寻找可利用知识，从而构造一条代价最小的推理路线，使问题得以解决的过程称为搜索。简单地说，搜索就是利用已知条件（知识）寻求解决问题办法的过程。

根据是否采用智能方法，搜索算法分为盲目搜索算法和智能搜索算法。

盲目搜索算法是指在问题求解过程中，不运用启发式知识，只按照一般的逻辑法则或控制性知识，在预定的控制策略下进行搜索，在搜索过程中获得的中间信息不用来改变控制策略。由于搜索总按照预先规定的线路进行，没有考虑问题本身的特性，这种方法缺乏对求解问题的针对性，需要进行全方位的搜索，而没有选择最优的搜索途径。因此，这种搜索具有盲目性、灵活性差、效率低、容易出现“组合爆炸”问题，且不便于复杂问题的求解。

智能搜索算法在问题求解的过程中，为了提高搜索效率，会运用与问题有关的启发式知识，即解决问题的策略、技巧、窍门等时间经验和知识，来指导搜索朝着最有希望的方向前进，加速问题求解过程并找到最优解。根据基于的搜索机理，这种算法可以分为多种类型，例如，基于搜索空间的状态空间搜索、与或树搜索、博弈树搜索、基于生物演化过程的进化搜索等算法。状态空间搜索是一种用状态空间来表示和求解问题的搜索算法。与或树搜索是一种用与或树表示和求解问题的搜索方法。博弈树是一种特殊的与或树，主要用于博弈过程的搜索。进化搜索是一种模拟自然界生物演化过程的随机搜索算法，其典型代表为遗传算法。

在搜索问题中，主要的工作是找到好的搜索算法。一般搜索算法的评价准则如下所述。

（1）完备性。如果存在一个解，该策略是否保证能找到？

（2）实践复杂性。需要多长时间可以找到解？

（3）空间复杂性。执行搜索需要多大的存储空间？

（4）最优性。如果存在不同的几个解，该策略是否可以发现最高质量的解？

3.2　状态空间搜索

在人工智能中，状态空间搜索是通过将问题生成状态空间对问题进行求解的。其基本思想：首先把问题的初始状态（即初始节点）作为当前状态，选择使用的算符对其进行操作。生成一组子状态（或称后继状态、后继节点、子节点），然后检查目标状态是否在其中出现。若出现，则搜索成功，找到问题的解；若不出现，则按某种搜索策略从已生成的状态中再选一个状态作为当前状态。重复上述过程，直到目标状态出现或不再有可供操作的状态及算法为止。

3.2.1　状态空间表示

人工智能解决问题的实质可以抽象为一个“问题求解”的过程。而该问题的实质是一个搜索的过程。状态空间表示法是用来描述搜索过程的一种常见方法。该方法把问题抽象为寻求初始节点到目标节点可行路径的问题。许多问题采用试探性搜索方法进行求解，即在某个可能的解空间内寻找一个解。这种基于解空间的问题表示和求解方法为状态空间表示法，该方法以状态（State）和操作（Operation）为基础来表示与求解问题。状态是为描述某类不同事物间的差别而引入的一组最少变量的有序集合，表示为 $Q=[q_0,q_1,\cdots,q_n]$，式中每个元素 q_i 称为状态变量，给定每个分量的一组值就能够得到一个具体的状态。操作也称为算符，对应过程型知识，即状态转换规则，是把问题从一种状态变为另一种状态的手段。它可以是一个机械步骤、一个运算、一条规则或一个过程。操作可理解为状态集合上的一个函数，它描述了状态之间的关系，通常表示为 $F=\{f_1,f_2,\cdots,f_m\}$。

问题的状态空间是一个表示该问题的全部可能状态及其关系的集合，包含问题初始状态集合 Q_s、操作符集合 F 及目标状态集合 Q_g，因此状态空间可用三元组 $\{Q_s,F,Q_g\}$ 描述。状态空间也可以用一个赋值的有向图来描述，称为状态空间图。在状态空间图中包含了操作和状态之间的转换关系，其中节点表示问题的状态，有向边（弧）表示操作。如果某条弧从节点 n_i 指向 n_j，那么节点 n_j 就为 n_i 的后继或后裔，n_i 为 n_j 的父辈或祖先，状态空间图如图 3-1 所示。某个节点序列（$n_{i,1},n_{i,2},\cdots,n_{i,k}$），当 j=2,3,$\cdots$,k 时，如果每一个 $n_{i,j-1}$ 都有一个后继节点 $n_{i,j}$ 存在，那么就把这个节点序列称作从节点 $n_{i,1}$ 到 $n_{i,k}$ 的长度为 k 的路径，其示意图如图 3-2 所示。如果从节点 n_i 到节点 n_j 存在一条路径，那么就称节点 n_j 是从节点 n_i 可达到的节点，或者称节点 n_j 为节点 n_i 的后裔。

状态空间表示法是从某个初始状态开始的，每次加一个操作符，递增地建立起操作符的实验序列，直到达到目标状态为止。寻找状态空间的全部过程包括从旧的状态描述产生新的状态描述，以及此后检验这些新的状态描述，看其是否描述了该目标状态。对于某些最优化问题，仅仅找到到达目标的任一路径是不够的，还必须找到按某个状态实现最优化的路径。

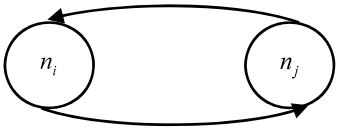


图 3-1　状态空间图

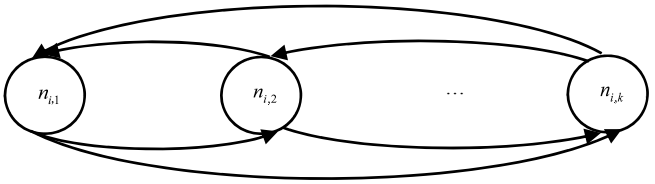


图 3-2　从节点 $n_{i,1}$ 到 $n_{i,k}$ 的长度为 k 的路径示意图

例 3-1　最短路径问题。一名推销员要去若干个城市推销商品，图 3-3 所示为城市间的距离。该推销员从某个城市出发，经过所有城市后，回到出发地。问题：应如何选择行进路径，使总的行程最短。

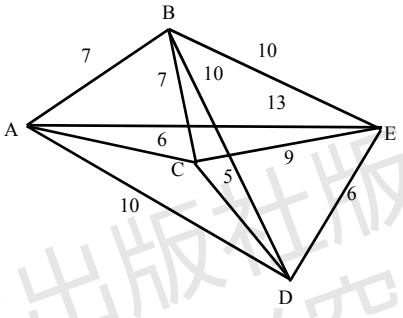


图 3-3　城市间的距离

解：推销员从 A 出发，可能到达 B、C、D 或者 E。到达 C 后，又可能到达 D 或 E，以此类推，图 3-4 所示为推销员的状态空间图。由图 3-4 可知，推销员应选择的最短路径为 ACDEBA。

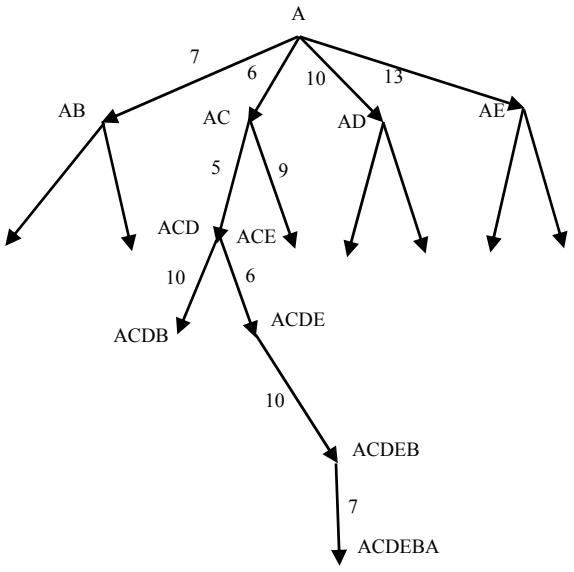


图 3-4　推销员的状态空间图

状态空间搜索时一般需要两种数据结构，即 OPEN 表和 CLOSED 表。OPEN 表用于存放刚生成的节点，这些节点也是带扩展的，对于不同的搜索策略，OPEN 表中节点的排放顺序也是不同的。CLOSED 表则用来存储将要扩展或已经扩展的节点，图 3-5 所示为 OPEN 表和 CLOSED 表的示例。

OPEN 表

节点	父节点编号

CLOSED 表

编号	节点	父节点编号

图 3-5　OPEN 表和 CLOSED 表的示例

基于状态空间的搜索策略分为盲目搜索策略和启发式搜索策略两大类。启发式搜索在搜索中要使用与问题有关的启发式信息，并以这些启发式信息指导搜索过程，可以有效地求解结构复杂的问题，具体内容将在后面进行详细介绍。盲目搜索不使用与问题有关的启发式信息，按规定的路线进行搜索，适用于其状态空间图是树状结构的一类问题。问题自身特性对搜索控制策略没有任何影响，从而使得盲目搜索带有盲目性、效率不高，不便于解决复杂的问题。盲目搜索有广度优先搜索、深度优先搜索和有界深度优先搜索三种方法。

广度优先搜索也称为宽度优先搜索，它沿着树的宽度遍历树的节点，其示意图如图 3-6 所示，其中标号代表节点搜索次序。其基本思想：从初始节点开始，逐层地对节点进行扩展并考虑它是否为目标节点，在第 n 层的节点没有全部扩展并考察完之前，不对第 n+1 层的节点进行扩展。OPEN 表中的节点总按进入的先后顺序进行排列，先进入的节点排在前面，后进入的节点排在后面。

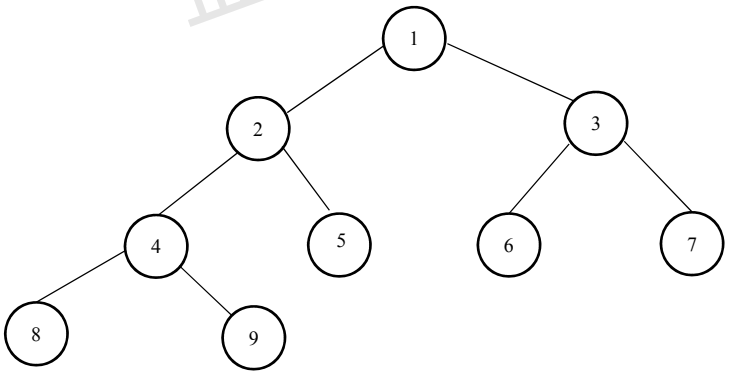


图 3-6　广度优先搜索示意图

深度优先搜索的特点是优先扩展最新产生的节点，其示意图如图 3-7 所示，标号为搜索的先后顺序。其基本思想：从初始节点开始，在其子节点中选择一个节点进行考察，若不是目标节点，则在该节点的子节点中选择一个节点进行考察，一直如此搜索下去。只有当到达某个子节点，且该节点既不是目标节点又不能扩展时，才选择其兄弟节点或先辈节点进行考察。

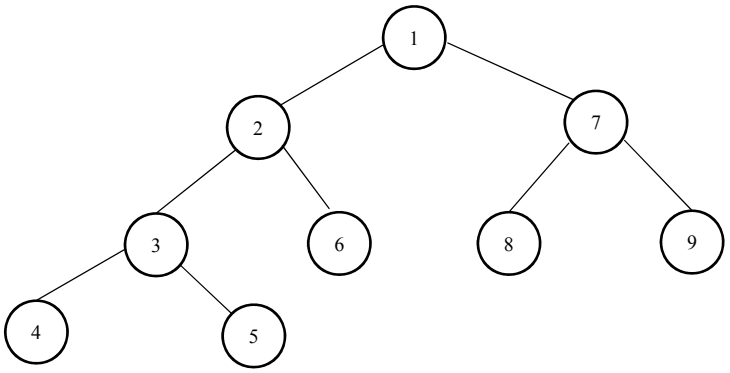


图 3-7　深度优先搜索示意图

深度优先搜索可能陷入无情分支的死循环而得不到解。为了解决深度优先搜索搜索不完备的问题，提出了有界深度优先搜索方法。有界深度优先搜索的基本思想：对深度优先搜索引入搜索深度的界限，当搜索深度达到深度界限，且尚未出现目标节点时，换一个分支进行搜索。

微课视频

3.2.2　启发式信息与估价函数

在启发式搜索过程中，关键是如何确定下一个要考察的节点，确定的方法不同就形成了不同的搜索策略。如果在确定节点时能充分利用与问题求解有关的特性信息，估计出节点的重要性，就能在搜索时选择重要性较高的节点，以利于求得最优解。像这样可用于指导搜索过程，且与具体问题求解有关的控制信息称为启发式信息。

用于估价节点重要性的函数称为估价函数，其一般形式为

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

式中，$g(n)$ 是代价函数，表示从初始节点 S_0 到节点 n 已经实际付出的代价；$h(n)$ 是启发式函数，表示从节点 n 到目标节点 S_g 的最优路径的估计代价。启发式函数 $h(n)$ 体现了问题的启发式信息，其形式要根据问题的特性确定。例如，$h(n)$ 可以是节点 n 到目标节点的距离，也可以是节点 n 处于最优路径上的概率等。

估价函数 $f(n)$ 表示从初始节点 S_0 经过节点 n 到达目标节点 S_g 的最优路径的代价估计值。它的作用是估价 OPEN 表中各节点的重要程度，决定它们在 OPEN 表中的次序。其中，代价函数 $g(n)$ 指出了搜索的横向趋势，它有利于搜索的完备性，但影响搜索的效率。如果只关心到达目标节点的路径，并且希望有较高的搜索效率，则 $g(n)$ 可以忽略，但此时会影响搜索的完备性。因此，在确定 $f(n)$ 时，要权衡各种利弊得失，使 $g(n)$ 与 $h(n)$ 各占适当的比重。

例 3-2　设有如下结构的移动奖牌游戏：

B	B	B	W	W	W	E

其中，B 代表黑色奖牌；W 代表白色奖牌；E 代表该位置为空。该游戏的玩法如下：

①当一个奖牌移入相邻的空位置时，费用为 1 个单位；②一个奖牌至多可跳过两个奖牌进入空位置，其费用等于跳过的奖牌数加 1。要求把所有的 B 都移至所有的 W 的右边，请设计启发式函数。

解：根据要求可知，W 左边的 B 越少越接近目标，因此可用 W 左边 B 的个数作为启发式函数，即

$$h(n)=3\times(\text{每个W左边B的个数的总和})$$

这里乘以系数 3 是为了扩大 $h(n)$ 在 $f(n)$ 中的比重。例如，对于

B	E	B	W	W	B	W

则有

$$h(n)=3\times(2+2+3)=21$$

例 3-3　八数码难题。设在 3×3 的方格棋盘上分别放置了 1～8 这 8 个数码，初始状态为 S_0，目标状态为 S_g，如图 3-8 所示。

S_0

2	8	3
1		4
7	6	4

S_g

1	2	3
8		4
7	6	4

图 3-8　八数码难题

若估价函数为

$$f(n)=d(n)+W(n) \tag{3-2}$$

式中，$d(n)$ 表示节点 n 在搜索树中的深度；$W(n)$ 表示节点 n 中“不在位”的数码个数。请计算初始状态 S_0 的估价函数值 $f(S_0)$。

解：在本例的估价函数中，取 $g(n)=d(n)$，$h(n)=W(n)$。它说明是用从 S_0 到 n 的路径上的单位代价来表示实际代价的，用 n 中“不在位”的数码个数作为启发式信息。一般来说，某节点中的“不在位”的数码个数越多，说明它离目标节点越远。

对于初始节点 S_0，由于 $d(S_0)=0$，$W(S_0)=3$，因此有 $f(S_0)=0+3=3$。

这个例子只是为了说明估价函数的含义及估价函数值的计算。在问题搜索过程中，除了需要计算初始节点的估价函数，更多的是要计算新生成节点的估价函数值。

3.2.3　A 算法

根据 3.2.1 节的内容可知，状态空间搜索通常需要用到 OPEN 表和 CLOSED 表两种数据结构。其中，OPEN 表用来存放未扩展的节点，CLOSED 表用来存放已扩展的节点。如果能在搜索的每步都利用优化函数 $f(n)=g(n)+h(n)$ 对 OPEN 表中的节点进行排序，则该搜索算法为 A 算法。由于估价函数中带有问题自身的启发式信息，因此 A 算法又称为启发式搜索算法。

根据搜索过程中要选择扩展节点的范围，启发式搜索算法可分为全局择优搜索算法和局部择优搜索算法。在全局择优搜索算法执行过程中，每当需要扩展节点时，总是从 OPEN 表的所有节点中选择一个估价函数值最小的节点进行扩展。在局部择优搜索算法执行过程中，每当需要扩展节点时，总是从刚生成的子节点中选择一个估价函数值最小的节点进行扩展。

全局择优搜索算法的搜索过程可描述如下。

（1）把初始节点 S_0 放入 OPEN 表中，$f(S_0)=g(S_0)+h(S_0)$。

（2）如果 OPEN 表为空，则问题无解，失败退出。

（3）把 OPEN 表的第一个节点取出放入 CLOSED 表，并记该节点为 n。

（4）考察节点 n 是否为目标节点。若是，则找到了问题的解，成功退出。

（5）若节点 n 不可扩展，则转至第（2）步。

（6）扩展节点 n，生成其子节点 $n_i(i=1,2,\cdots)$，计算每个子节点的估价函数值 $f(n_i)$ $(i=1,2,\cdots)$，并为每个子节点设置指向父节点的指针，然后将它们放入 OPEN 表。

（7）根据各节点的估价函数值，对 OPEN 表中的全部节点按从小到大的顺序，重新进行排序。

（8）转至第（2）步，继续执行。

由于上述算法的第（7）步要对 OPEN 表中的全部节点按其估价函数值从小到大重新进行排序，这样在算法第（3）步中取出的节点一定是 OPEN 表的所有节点中估价函数值最小的。因此，它是一种全局择优的搜索算法。

对上述算法进一步分析还可以发现：如果取估价函数 $f(n)=g(n)$，则它退化成代价树的广度优先搜索（在后续章节中介绍）；如果取估价函数 $f(n)=h(n)$，则它退化为广度优先搜索。可见，广度优先搜索和代价树的广度优先搜索是全局择优搜索的两个特例。

局部择优搜索算法的搜索过程可描述如下。

（1）把初始节点 S_0 放入 OPEN 表中，$f(S_0)=g(S_0)+h(S_0)$。

（2）如果 OPEN 表为空，则问题无解，失败退出。

（3）把 OPEN 表的第一个节点取出放入 CLOSED 表，并记该节点为 n。

（4）考察节点 n 是否为目标节点。若是，则找到了问题的解，成功退出。

（5）若节点 n 不可扩展，则转至第（2）步。

（6）扩展节点 n，生成其子节点 $n_i(i=1,2,\cdots)$，计算每个子节点的估价函数值 $f(n_i)$ $(i=1,2,\cdots)$，并按估价函数值从小到大的顺序依次将子节点放到 OPEN 表中。

（7）为每个子节点设置指向父节点的指针。

（8）转至第（2）步，继续执行。

比较全局择优搜索与局部择优搜索的搜索过程可以看出，它们的区别仅在第（6）步和第（7）步。

例 3-4　八数码难题。设问题的初始状态为 S_0 和目标状态为 S_g，如图 3-8 所示，估价函数与例 3-3 的相同。请用全局择优搜索解决该问题。

解：这个难题的全局择优搜索树如图 3-9 所示，每个节点旁边的数字是该节点的估价

函数值。例如，对于节点 S_2，其估价函数值的计算为

$$f(S_2)=d(S_2)+W(S_2)=2+2=4$$

从图 3-9 还可以看出，该问题的解为

$$S_0 \to S_1 \to S_2 \to S_3 \to S_g$$

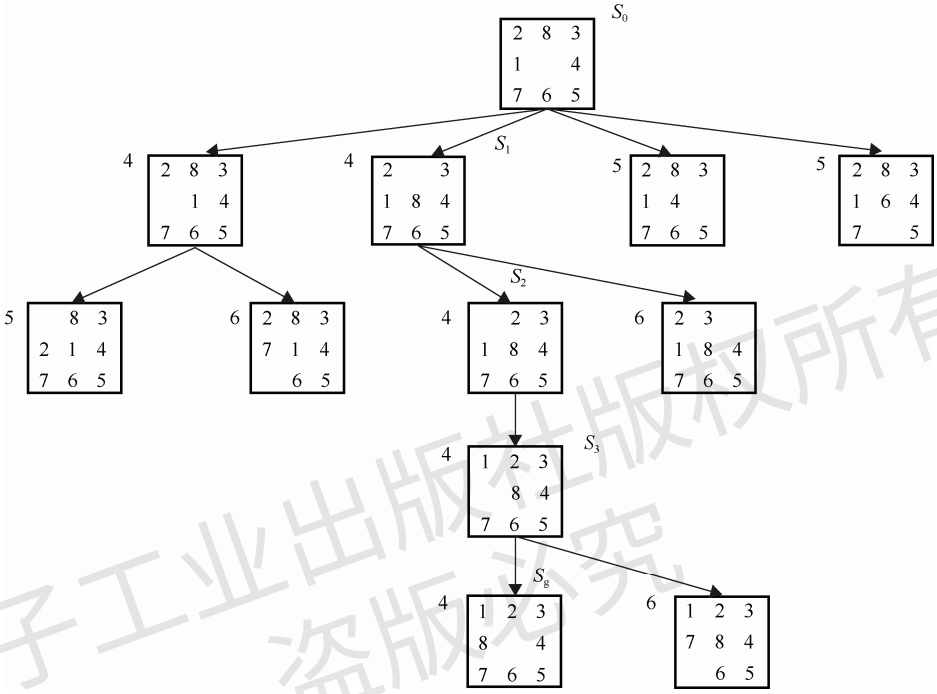


图 3-9　八数码难题的全局择优搜索树

3.2.4　A*算法

A 算法没有对估价函数 $f(n)$ 做任何限制。实际上，估价函数对搜索过程十分重要，如果选择不当，则有可能找不到问题的解，或者找到的不是问题的最优解。为此，需要对估价函数进行某些限制。A*算法就是对估价函数加上一些限制后得到的一种启发式搜索算法。

满足以下条件的搜索过程称为 A*算法。

（1）把 OPEN 表中的节点按估价函数

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

的值从小到大进行排序。

（2） $g(n)$ 是对 $g^*(n)$ 的估计，且 $g(n)>0$， $g^*(n)$ 是从初始节点 S_0 到节点 x 的最小代价。

（3） $h(n)$ 是 $h^*(n)$ 的下界，即对所有的节点 x 均有

$$h(n) \leqslant h^*(n) \tag{3-4}$$

式中， $h^*(n)$ 是从节点到目标节点的最小代价，若有多个目标节点，则为其中最小的一个。

在 A*算法中， $g(n)$ 比较容易得到，它实际上就是从初始节点 S_0 到节点 n 的路径代价，恒有 $g(n) \geqslant g^*(n)$；而且在算法执行过程中随着更多搜索信息的获得， $g(n)$ 的值呈下降趋

势。例如，在图 3-10 中，从节点 S_0 开始，经扩展得到 n_1 与 n_2，且 $g(n_1)$=3，$g(n_2)$=7。对 n_1 扩展后得到 n_2 和 n_3，此时 $g(n_2)$=6，$g(n_3)$=5。显然，后来算出的 $g(n_2)$ 比先前算出的小。

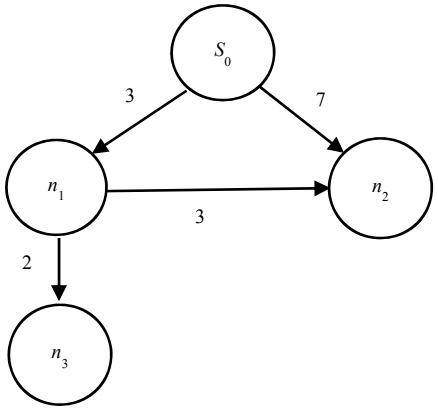


图 3-10　$g(n)$ 的计算

启发式函数 $h(n)$ 的确定依赖于具体问题领域的启发式信息，其中，$h(n) \leqslant h^*(n)$ 的限制是十分重要的，它可保证 A*算法找到最优解。

A*算法的主要特点如下。

（1）A*算法是可纳的。一个算法是可纳的，即如果存在解，则一定能找到最优的解。对于可解状态空间图（即从初始节点到目标节点有路径存在）来说，如果一个搜索算法能在有限步内终止并且能找到最优解，则称该搜索算法是可纳的。

（2）A*算法是可纳的，即它能在有限步内终止并找到最优解。

（3）A*算法是完备的。一个算法是完备的，即如果存在解，则它一定能找到该解并结束。

（4）A*算法的最优性。

（5）A*算法的搜索效率在很大程度上取决于 $h(n)$，在满足 $h(n) \leqslant h^*(n)$ 的前提下，$h(n)$ 的值越大越好。启发式函数 $h(n)$ 的值越大，标明它携带的启发式信息越多，搜索时扩展的节点越少，搜索的效率越高。

（6）启发式函数的单调性限制。

（7）在 A*算法中，每当要扩展一个节点时都要先检查其子节点是否已在 OPEN 表或者 CLOSED 表中，有时还需要调整指向父节点的指针，这就增加了搜索的代价。如果对启发式函数 $h(n)$ 加上单调性限制，就可以减少检查及调整的工作量，从而降低搜索代价。

（8）所谓单调限制是指 $h(n)$ 满足如下两个条件。

① $h(S_g)=0$，S_g 是目标节点。

② 设 x_j 是节点 x_i 的任意子节点，则有

$$h(x_j)-h(x_i) \leqslant c(x_i,x_j) \tag{3-5}$$

式中，$c(x_i,x_j)$ 是节点 x_i 到节点 x_j 的边代价。

3.3　与或树搜索

与或树搜索，也称为与或图搜索，其搜索策略是确定节点是可解节点或不可解节点。一般情况下会循环用到两个过程，即可解表示过程和不可解标识过程，这两个过程都是自下向上进行的，由子节点的可解性确定父节点、祖父节点等的可解性。

3.3.1　与或树表示


微课视频

根据问题规约法的基本原理，对于一个复杂问题，可以把此问题分解成若干子问题。如果把每个子问题都解决了，整个问题也就解决了。如果子问题不容易解决，还可以再分成子问题，直至所有的子问题都解决了，则这些子问题的解的组合就构成了整个问题的解。与或树是用于表示此类求解过程的一种方法，是一种图或树的形式，基于的是人们在求解问题时的一种思维方法。

（1）分解：与树。一个复杂问题 P 分解为与之等价的一组简单的子问题 $P_1,P_2,\cdots,P_n$，子问题还可分为更小、更简单的子问题，以此类推。当这些子问题全都解决时，原问题 P 也就解决了；任何一个子问题 $P_i\ (i=1,2,\cdots,n)$ 无解，都将导致原问题 P 无解。这样的问题与这组子问题之间形成了“与”的逻辑关系。这个分解过程可用一个有向图表示：问题和子问题都用相应的节点表示，从问题 P 到每个子问题 P_i 都只用一个有向边连接，然后用一段弧将这些有向边连接起来，以标明它们之间存在的与的关系。这种有向图称为与图或者与树。

（2）等价变换：或树。把一个复杂的问题 P 等价变换为与之等价的一组简单的子问题 $P_1,P_2,\cdots,P_n$，而子问题还可再等价变换为若干更小、更简单的子问题，以此类推。当这些子问题中有任何一个子问题 $P_i\ (i=1,2,\cdots,n)$ 有解时，原问题 P 就解决了；只有当全部子问题无解时，原问题 P 才无解。这样，问题与这组子问题形成了“或”的关系。这个等价变换同样可用一个有向图来表示，这种有向图称为或图或者或树。表示方法类似与图的表示，只是在或图中不用弧将有向边连接起来。

（3）与或树。在实际问题求解过程中，常常既有分解又有等价变换，因而常将两种图结合起来一同用于表示问题的求解过程。此时，所形成的图就称为与或图或者与或树。

可以把与或树视为对一般树（或图）的扩展；或反之，把一般树视为与或树的特例，即一般树不允许节点间具有“与”关系，所以又可把一般树称为或树。与一般树类似，与或树也有根节点，用于指示初始状态。由于同父的子节点间可以存在“与”关系，父、子节点间不能简单地以弧线关联，因此需要引入“超连接”概念。同样的道理，在典型的与或树中，解路径往往不复存在，代之以广义的解路径——解树。

图 3-11 给出了一个抽象的与或树简例，节点的状态描述不再显式给出。

下面基于该简例引入和解释与或树搜索的基本概念。

（1）K-连接。K-连接用于表示从父节点到子节点间的连接，也称为父节点的外向连接，并以圆弧指示同父子节点间的“与”关系，K 为这些子节点的个数。一个父节点可以有多

个外向的 K-连接。例如，根节点 n_0 有两个 K-连接：一个 2-连接指向子节点 n_1和n_2，一个 3-连接指向子节点 n_3、n_4 和 n_5。$K>1$的连接也称为超连接；K=1时，超连接蜕化为普通连接；而当所有超连接的 K 都等于 1 时，与或树蜕化为一般树。

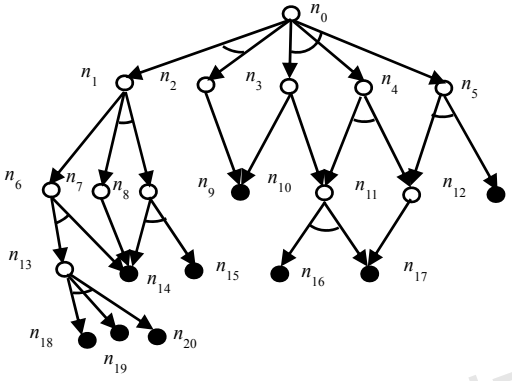


图 3-11　与或树简例

（2）根、叶、终节点。无父节点的节点称为根节点，用于指示问题的初始状态；无子节点的节点称为叶节点。由于问题规约伴随着问题分解，因此目标状态不再由单一节点表示，而应由一组节点联合表示。能用于联合表示目标状态的节点称为终节点；终节点必定是叶节点，反之不然；非终节点的叶节点往往指示了解搜索的失败。

（3）解树的生成。在与或树搜索过程中，可以这样建立解树：自根节点开始选择一个外向连接，并从该连接指向的每个子节点出发，再选择一个外向连接，如此反复，直到所有外向连接都指向终节点为止。例如，从图 3-12 所示的与或树根节点 n_0 开始，选左边的 K=2 的外向连接，指向节点 n_1和n_2，再从 n_1、n_2 分别选外向连接；从 n_1 选左边的 K=1 的外向连接，指向 n_6，依次进行，直至终节点 n_{14}、n_{18}、n_{19}和n_{20}；从 n_2 只有一个 K=1 的外向连接指向终节点 n_9，从而，生成了图 3-12（a）所示的一个解树。注意，解树是遵从问题规约策略而搜索到的，解图中不存在节点或节点组之间的“或”关系；换言之，解树纯粹是一种“与”树。另外，正因为与或树中存在“或”关系，所以往往会搜索到多个解树，本例中就有 4 个，如图 3-12 所示。

为了确保在与或树中搜索解树的有效性，要求解树是无环的，即任何节点的外向连接均不得指向自己或自己的先辈，否则会使搜索陷入死循环，换言之，会导致解树有环的外向连接不能选用。下面给出解树、解树代价、能解节点和不能解节点的定义。

（1）解树：与或树（记为 G）中任一节点（记为 n）到终节点集合的解树（记为 G'）是 G 的子树。

若 n 是终节点，则 G' 就由单一节点 n 构成。

若 n 有一外向 K-连接指向子节点 $n_1,n_2,\cdots,n_k$，且每个子节点都有到终节点集合的解树，则 G 由该 K-连接和所有子节点的解树构成。

否则，不存在 n 到终节点集合的解树。

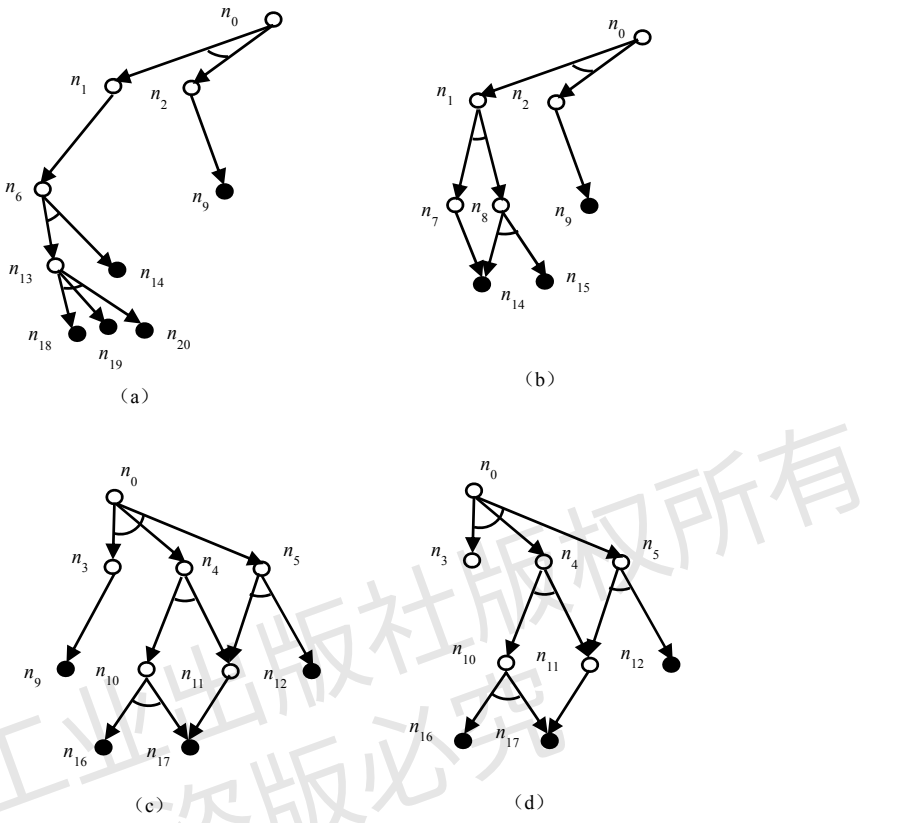


图 3-12　4 个可能的解树

（2）解树代价：以 $C(n)$ 指示节点 n 到终节点集合解图的代价，并令 K-连接的代价为 K。

（3）能解节点。

终节点是能解节点。

若节点 n 有一外向 G-连接指向子节点 $n_1, n_2, \cdots, n_k$，且这些子节点都是能解节点，则 n 是能解节点。

（4）不能解节点。

非终节点的叶节点是不能解节点。

若节点 n 的每个外向连接都至少指向一个不能解节点，则 n 是不能解节点。

能解节点和不能解节点如图 3-13 所示。

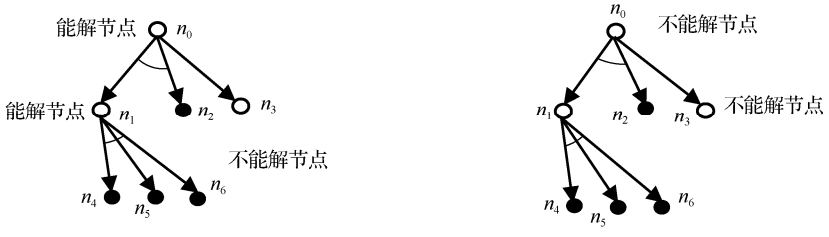


图 3-13　能解节点和不能解节点

基于与或树的搜索策略分为盲目搜索策略和启发式搜索策略两大类。盲目搜索策略包括广度优先搜索策略和深度优先搜索策略。搜索从初始节点开始，先自上而下进行搜索，寻找终节点及端节点，然后自下而上地进行标识，一旦初始节点被标识为可解节点或不可解节点，搜索就不再进行。这两种搜索策略都是按确定路线进行的，当要选择一个节点进行扩展时，由于只考虑了节点在与或树中所处的位置，而没考虑要付出的代价，因而求得的解树不一定是代价最小的，即不一定是最优解树。与或树的盲目搜索策略与状态空间的盲目搜索策略类似，与或树的广度优先搜索按照“先产生的节点先扩展”的原则进行搜索，在整个搜索过程中多次调用可解标识过程和不可解标识过程。与或树的深度优先搜索过程和状态空间的深度优先搜索过程基本相同，只是将扩展节点的子节点放入 OPEN 表首部，并为每个子节点配置指向父节点的指针，并且与状态空间的有界深度搜索类似，与或树的深度优先搜索也可以规定一个深度界限，使其在规定范围内进行搜索。启发式搜索策略的具体内容将在本章后续小节中详细介绍。

3.3.2 解树的代价

与或树的启发式搜索算法是利用搜索过程所得到的启发式信息寻找最优解树的过程。对搜索的每一步，算法都试图找到一个最有希望成为最优解树的子树。最优解树是指代价最小的那棵解树。这里首先讨论什么是解树的代价。

要寻找最优解树，首先需要计算解树的代价。在与或树的启发式搜索过程中，解树的代价可按如下规则进行计算。

（1）若 n 是终节点，则其代价 $h(n)=0$。

（2）若 n 为或节点，且子节点为 $n_1,n_2,\cdots,n_k$，则 n 的代价为

$$h(n)=\min_{1\leqslant i\leqslant k}\left[c(n,n_i)+h(n_i)\right] \tag{3-6}$$

式中，$c(n,n_i)$ 是节点 n 到其子节点 n_i 的边代价。

（3）若 n 为与节点，且子节点为 $n_1,n_2,\cdots,n_k$，则 n 的代价可用和代价法或最大代价法计算。和代价法的计算公式为

$$h(n)=\sum_{i=1}^{k}\left[c(n,n_i)+h(n_i)\right] \tag{3-7}$$

最大代价法的计算公式为

$$h(n)=\min_{1\leqslant i\leqslant k}\left[c(n,n_i)+h(n_i)\right] \tag{3-8}$$

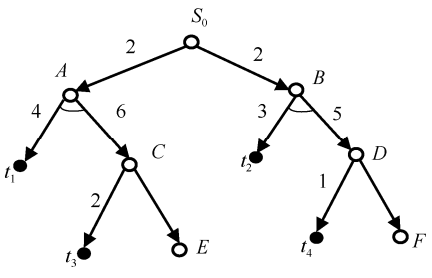


图 3-14　与或树的代价

（4）若是端节点，但不是终节点，则 n 不可扩展，其代价定义为 $h(n)=\infty$。

（5）根节点的代价即解树的代价。

例 3-5　设图 3-14 所示的是一棵与或树，其中包括两棵解树，左边的解树由 S_0、A、t_1、C 及 t_3 组成，右边的解树由 S_0、B、t_2、D 及 t_4 组成。在此与或树中，t_1、t_2、t_3、t_4 为终节点，E、F 是端节点，边上的数字是该边的代价。请计算解树的代价。

解：先计算左边的解树。

按和代价计算：$h(S_0)=2+4+6+2=14$

按最大代价计算：$h(S_0)=8+2=10$

再计算右边的解树。

按和代价计算：$h(S_0)=1+5+3+2=11$

按最大代价计算：$h(S_0)=6+2=8$

在本例中，无论是按和代价还是最大代价计算，右边的解树都是最优解。但在有些情况下，当采用的代价法不同时，找到的最优解树也有可能不同。

3.3.3　与或树的有序搜索

与或树的有序搜索是一种启发式搜索，它的目的是求出最优解树，即代价最小的解树。为了找到最优解树，搜索过程的任何时刻都应该选择那些最有希望成为最优解树的一部分的节点进行扩展。由于这些节点及其父节点所构成的与或树最有可能成为最优解树的一部分，因此称它为希望解树，简称希望树。应当注意的是，希望树是会随搜索过程而不断变化的。下面给出希望树的定义。

希望树 T：

（1）初始节点 S_0 在希望树 T 中。

（2）如果 n 是具有子节点 $n_1,n_2,\cdots,n_k$ 的或节点，则 n 的某个子节点 n_i 在希望树 T 中的充分必要条件是

$$\min_{1\leqslant i\leqslant k}\left[c(n,n_i)+h(n_i)\right] \tag{3-9}$$

（3）如果 n 是与节点，则 n 的全部子节点都在希望树 T 中。

在搜索的过程中，随着新节点不断生成，节点的价值是在不断变化的，因此希望树也是在不断变化的。在某一时刻，这一部分节点构成希望树，但在另一时刻，可能是另一些节点构成希望树，随当时的情况而定。但不管如何变化，任一时刻的希望树都必须包含初始节点 S_0，而且它是对最优解树近根部的某种估计。

与或树的有序搜索是一个不断选择、修正希望树的过程。如果问题有解，则经过有序搜索将找到最优解树。搜索过程如下所述。

（1）把初始节点 S_0 放入 OPEN 表。

（2）求出希望树 T，即根据当前搜索树中节点的代价求出以 S_0 为根的希望树 T。

（3）依次把 OPEN 表中 T 的端节点 n 选出，并放入 CLOSED 表。

（4）如果节点 n 是终节点，则做如下工作。

① 标识节点 n 为可解节点。

② 对 T 应用可解标识过程，将 n 的先辈节点中的可解节点都标识为可解节点。

③ 若初始节点 S_0 能被标识为可解节点，则 T 就是最优解树，成功退出。

④ 否则，从 OPEN 表中删除具有可解性的先辈的所有节点。

（5）如果节点 n 不是终节点，且它不可扩展，则做如下工作。

① 标识节点 n 为不可解节点。

② 对 T 应用不可解标识过程，将 n 的先辈节点中的不可解节点都标识为不可解节点。

③ 若初始节点 S_0 能被标识为不可解节点，则失败退出。

④ 否则，从 OPEN 表中删除具有不可解性的先辈的所有节点。

（6）如果节点 n 不是终节点，但它可扩展，则做如下工作。

① 扩展节点 n，产生 n 的所有子节点。

② 把这些子节点都放入 OPEN 表，并为每个子节点配置指向父节点（节点 n）的指针。

③ 计算这些子节点的代价值及其先辈节点的代价值。

（7）转至第（2）步。

与或树的有序搜索算法的流程图如图 3-15 所示。

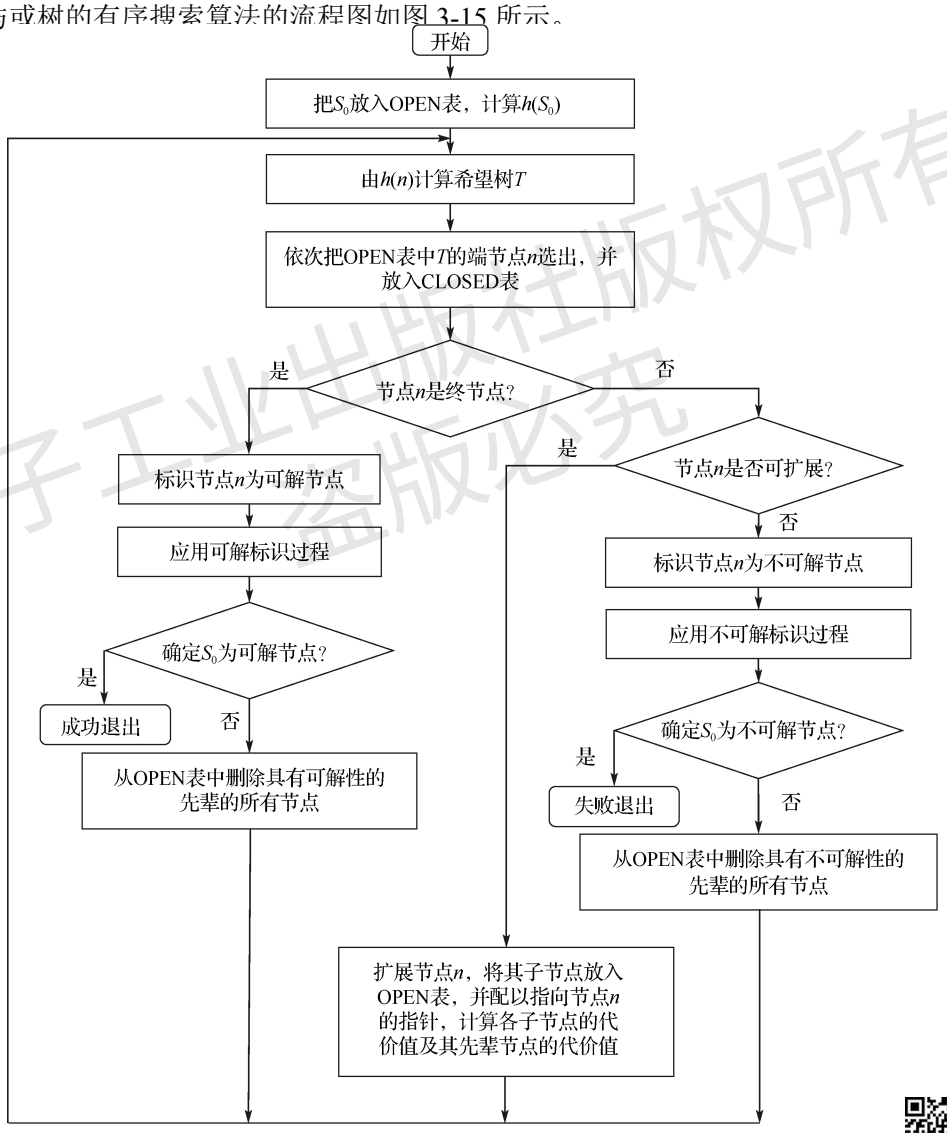


图 3-15　与或树的有序搜索算法的流程图

例 3-6　假设初始节点为 S_0，与或树每次扩展两层，并且一层是与节点，

微课视频

另一层是或节点。S_0 经扩展后得到图 3-16 所示的有待扩展的与或树，其中子节点 B、C、E、F 用启发式函数算出的 h 值分别为 $h(B)=3$、$h(C)=3$、$h(E)=3$、$h(F)=2$，假设每个节点到其子节点的代价为 1，即每条边的代价按 1 计算。

解：若按和代价计算，则有

$$h(A)=8，h(D)=7，h(S_0)=8$$

此时，S_0 右子树为希望树，因此对希望树的端点进行扩展。

假设对节点 E 扩展两层后的与或树如图 3-17 所示，节点旁的数为用启发式函数估算出的代价值。按照和代价计算可以得到：

$$h(G)=7，h(H)=6，h(E)=7，h(D)=11$$

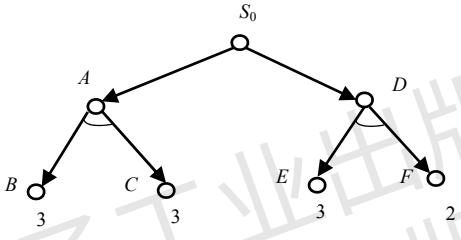


图 3-16　有待扩展的与或树

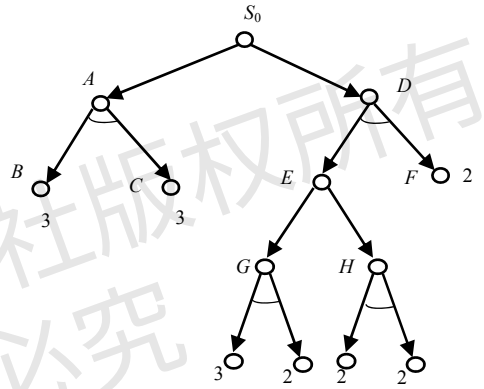


图 3-17　对节点 E 扩展两层后的与或树

此时，S_0 的右子树算出 $h(S_0)=12$。但是，由于 S_0 的左子树算出 $h(S_0)=9$，相比之下左子树的代价更小，所以改取左子树作为希望树，并对节点 B 进行扩展。

假设对节点 B 扩展两层后的与或树如图 3-18 所示，因为节点 L 的两个子节点是终节点，则按照和代价计算可以得到：

$$h(L)=2，h(M)=6，h(B)=3，h(A)=8$$

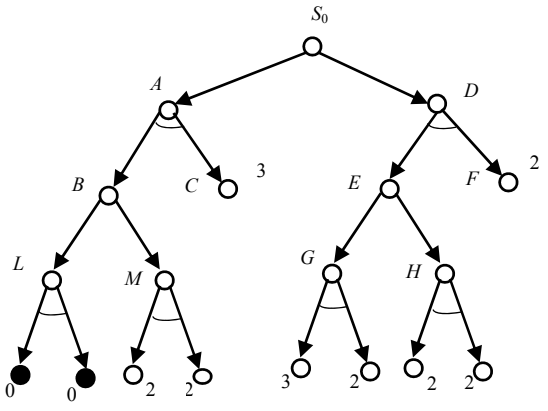


图 3-18　对节点 B 扩展两层后的与或树

由于节点 L 的两个子节点都是可解节点，所以节点 L、B 都是可解节点，但节点 C 不能确定为可解节点，所以节点 A 和 S_0 也不能确定为可解节点，所以对节点 C 进行扩展。

假设对节点 C 扩展两层后的与或树如图 3-19 所示，因为节点 N 的两个子节点是终节点，则按照和代价计算可以得到：

$$h(N)=2，h(P)=7，h(C)=3，h(A)=8$$

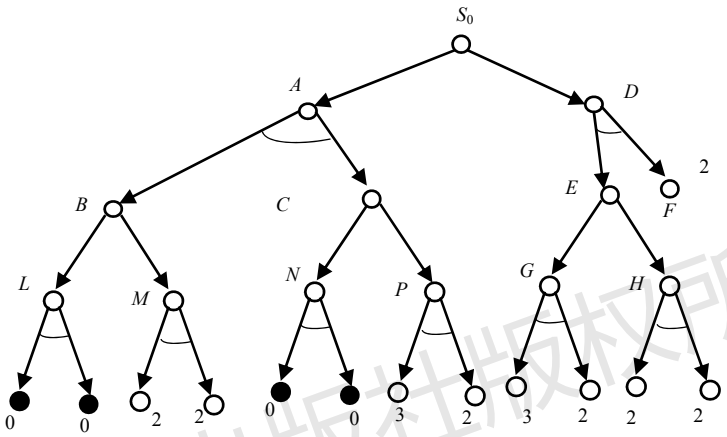


图 3-19　对节点 C 扩展两层后的与或树

由此推算，$h(S_0)=9$，最优解树由节点 S_0、A、B、C、L、N 构成。

3.4　博弈

广义的博弈涉及人类各方面的对策问题，如军事冲突、政治斗争、经济竞争等。博弈提供了一个可构造的任务领域，在这个领域中具有明确的胜利与失败。同样，博弈问题对人工智能研究提出了严峻的挑战，如如何表示博弈问题的状态、博弈过程和博弈知识等。所以，在人工智能领域中，通过计算机下棋等研究博弈的规律、策略和方法是具有实际意义的。本节将对二人博弈进行介绍。

3.4.1　博弈树

博弈是人们生活中常见的一种活动。下棋、打牌等活动均是博弈活动。博弈活动中一般有对立的几方，每一方都试图使自己的利益最大化。博弈活动的整个过程其实就是一个动态的搜索过程。本节介绍的博弈是二人博弈，具有二人零和、全信息、非偶然的特点，博弈双方的利益是完全对立的，从规则上容易得出：

（1）对垒的双方 MAX 和 MIN 轮流采取行动。

（2）博弈的结果只能有三种情况，即 MAX 胜，MIN 败；MAX 败，MIN 胜；和局。

（3）在博弈过程中，任何一方都了解当前的格局和过去的历史。

（4）任何一方采取行动前都根据当前的实际情况，进行得失分析，选择对自己最为有利而对对方最不利的对策，不存在“碰运气”的偶然因素，即双方都理智地决定自己的行为。

另外一种博弈是机遇性博弈，是指不可预测性的博弈，如掷币游戏等。对于机遇性博弈，由于不具备完备信息，所以在此不做讨论。

先来看一个例子，假设有七枚钱币，任一选手只能将已分好的一堆钱币分成两堆个数不等的钱币，两位选手轮流进行，直到每一堆都只有一个或两个钱币，不能再分为止，哪个选手遇到不能再分的情况，则为输。

用数字序列加上一个说明表示一种状态，其中数字表示不同堆中钱币的个数，说明表示下一步由谁来分，如（7，MIN）表示只有一个由七枚钱币组成的堆，由 MIN 来分，MIN 有三种可供选择的分法，即（6，1，MAX）、（5，2，MAX）、（4，3，MAX），其中 MAX 表示另一选手，不论哪一种方法，MAX 在它的基础上再做符合要求的划分，整个过程如图 3-20 所示。

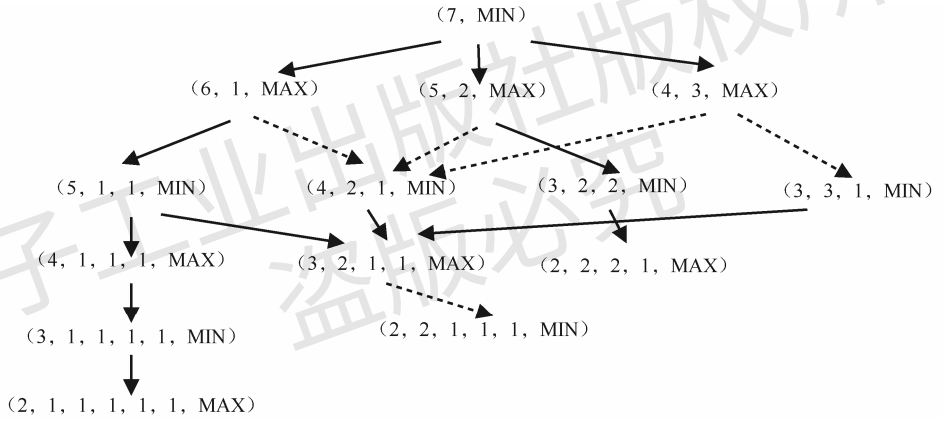


图 3-20　分钱币博弈的整个过程

在图 3-20 中已将双方可能的分法完全表示出来了，而且从中可以看出，无论 MIN 开始时怎么分，MAX 总可以获胜，取胜的策略用虚线表示。

实际的情况没有这么简单，任何一种棋都不可能将所有情况列尽，因此，只能模拟人“向前看几步”，然后做出决策，决定自己走哪一步最有利，也就是说，只能给出几层走法，然后按照一定的估算方法，决定走哪一步。

在双人完备信息博弈过程中，双方都希望自己能够获胜。因此当一方走步时，都是选择对自己最有利，而对对方最不利的走法。假设博弈双方为 MAX 和 MIN。在博弈的每一步，可供他们选择的方案都有很多种。从 MAX 的观点看，可供自己选择的方案之间是“或”关系，原因是主动权在自己手里，选择哪个方案完全由自己决定；而对那些可供 MIN 选择的方案之间是“与”的关系，这是因为主动权在 MIN 手中，任何一个方案都可能被 MIN 选中，MAX 必须防止那种对自己最不利的情况出现。

图 3-20 把双人博弈过程用图的形式表示了出来，这样就可以得到一棵与或树，这种与或树称为博弈树。在博弈树中，那些下一步该 MAX 走的节点称为 MAX 节点，而下一步该

MIN 走的节点称为 MIN 节点。博弈树具有如下特点。

（1）博弈的初始状态是初始节点。

（2）博弈树的与节点和或节点是逐层交替出现的。自己一方扩展的节点之间是或关系，对方扩展的节点之间是与关系。双方轮流扩展节点。

（3）整个博弈过程始终站在某一方的立场上，所以能使自己一方获胜的终局都是本原题，相应的节点也是可解节点，所有使对方获胜的节点都是不可解节点。

3.4.2 极大极小过程

在二人博弈问题中，为了从众多可选择的方案中选出一种对自己有利的方案，就要对当前情况以及将要发生的情况进行分析，从中选择最优者。最常见的分析方法是极大极小过程分析方法。假设博弈双方分别为 MAX 和 MIN，极大极小过程的基本思想如下。

（1）目的是为博弈双方中的一方（MAX 方）找到一种最优方案。

（2）计算当前所有可能的方案并进行比较，从而找到最优方案。

（3）方案的比较是指根据问题的特性定义一个估价函数，用来估算当前博弈树端节点的得分。此时估算出来的得分称为静态估值。对于静态估价函数，有如下规定：

① 有利于 MAX 方的态势，静态估值取值为正值。

② 有利于 MIN 方的态势，静态估值取值为负值。

③ 态势均衡的时候，静态估值取 0。

（4）当端节点的估值计算出来以后，再推算出其父节点的得分。推算的方法：对于或节点，选择其子节点中一个最大的得分作为父节点的得分，这是为了使自己在可供选择的方案中选一种对自己最有利的方案。对于与节点，选择其子节点中一个最小的得分作为父节点的得分，这是为了考虑最坏情况。这样计算出的父节点的得分称为倒推值。倒推值计算示例如图 3-21 所示，用□表示 MAX，○表示 MIN，端节点上的数字表示它对应的估价函数的值。

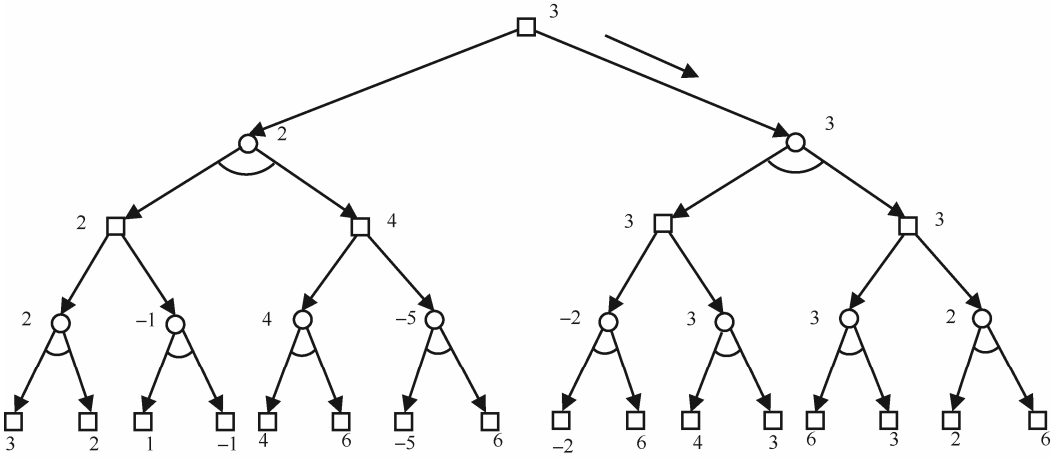


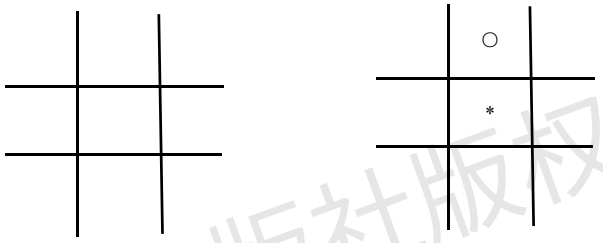
图 3-21　倒推值计算示例

（5）如果一种方案能获得较大的倒推值，则它就是当前最好的方案。

应当注意的是，当轮到 MIN 走步的节点时，MAX 应考虑最坏的情况，静态估值取极小值；当轮到 MAX 走步的节点时，MAX 应考虑最好的情况，静态估值取极大值。

在博弈问题中，每一个格局可供选择的方案有很多，因此会生成十分庞大的博弈树。试图利用完整的博弈树来进行极大极小过程分析是困难的，可行的方法是只生成一定深度的博弈树，然后进行极大极小过程分析，找出当前最好的方案。

例 3-7　一字棋游戏。设有如图 3-22（a）所示的 9 个空格。MAX 和 MIN 二人博弈，轮到谁走棋就往空格上放自己的一个棋子。谁先使自己的三个棋子串成一条直线，谁就取得胜利。



（a）　　（b）

图 3-22　一字棋

解：设 MAX 的棋子用*表示，MIN 的棋子用○表示。为了不至于生成太大的博弈树，假设每次仅扩展两层。设棋局为 p，估价函数 $e(p)$ 规定如下。

（1）若 p 为 MAX 必胜的棋局，则 $e(p)=+\infty$。

（2）若 p 为 MIN 必胜的棋局，则 $e(p)=-\infty$。

（3）若 p 为胜负未定的棋局，则 $e(p)=e(+p)-e(-p)$。

其中，$e(+p)$ 表示棋局 p 上所有空格都放上 MAX 的棋子*之后，三个*串成一条直线的总数目；$e(-p)$ 表示棋局 p 上所有空格都放上 MIN 的棋子○之后，三个○串成一条直线的总数目。因此，若 p 为图 3-22（b）所示的棋局，就有 $e(p)=6-4=2$。

另外，假定具有对称性的两个棋局是相同的，将二者视为同一个棋局。还假定 MAX 先走，我们站在 MAX 的立场上。

图 3-23 给出了 MAX 第一步走棋生成的博弈树，图中节点旁的数字表示相应节点的静态估值或推导值。由于*放在中间位置有最大的倒推值，故 MAX 第一步应选择 S_3。当 MAX 走 S_3 这一步棋后，MIN 的最优选择是 S_4。因为这一步棋的静态估值最小，对 MAX 最不利。不管 MIN 选择 S_4 还是 S_5，MAX 都将再次运用极大极小过程分析产生深度为 2 的博弈树，以决定下一步应该如何走棋，其过程与上面类似，这里不再赘述。

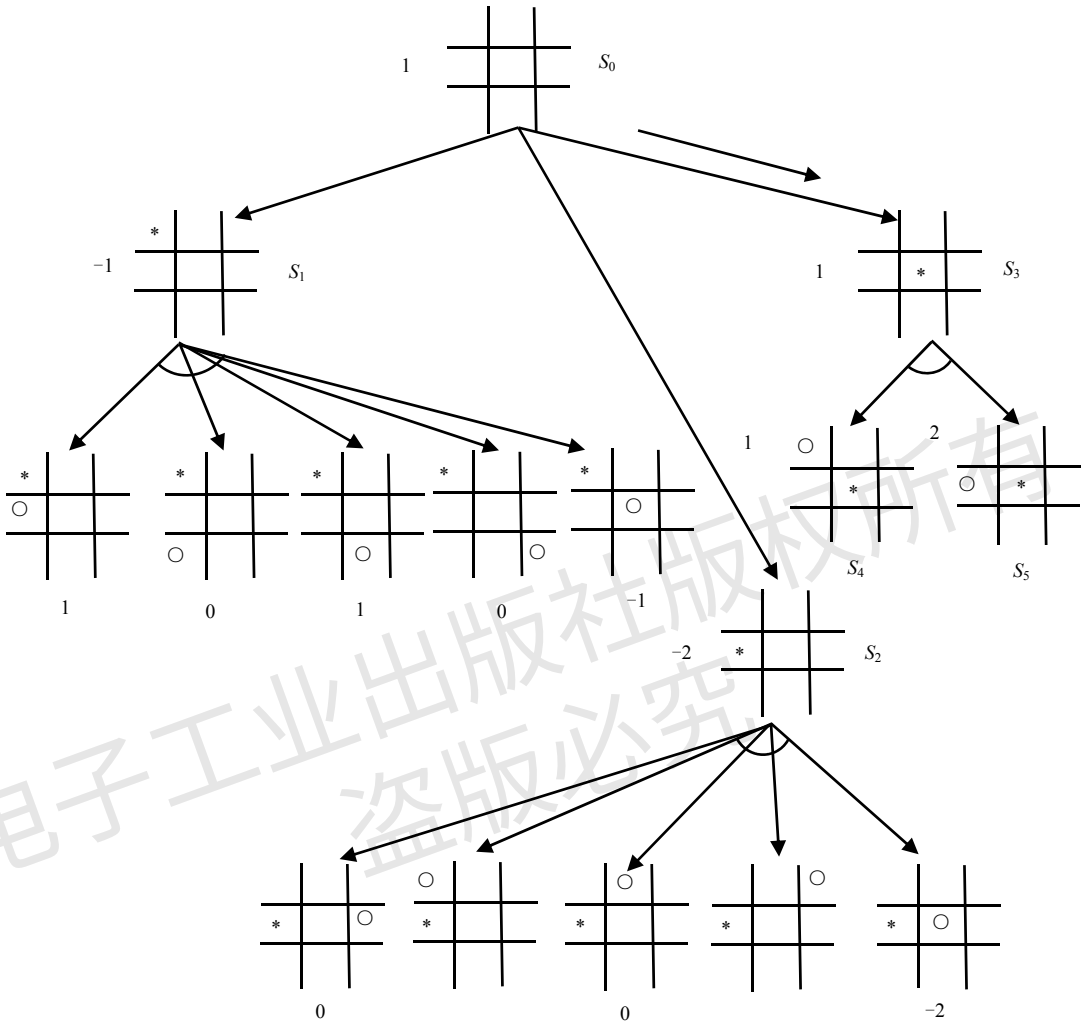


图 3-23　一字棋博弈的极大极小过程

3.4.3　α-β 过程

上面讨论的极大极小过程首先会生成一棵博弈树，而且会生成规定深度的所有节点。然后进行估值的倒推计算，这样使生成博弈树和估值的倒推计算两个过程完全分离，因此搜索效率较低。如果能边生成博弈树，边进行估值的倒推计算，则可能不必生成规定深度内的所有节点，以减少搜索的次数，这就是本节要讨论的 α-β 过程。

α-β 过程把生成后继节点和倒推计算结合起来，及时剪掉一些无用分支，以此来提高算法的效率。下面仍然以一字棋进行说明。现将图 3-23 所示的左边一部分画在图 3-24 中。

前面的过程实际上类似于宽度优先搜索，将每层格局均生成，现在用深度优先搜索来处理，比如在节点 A 处，若已生成 5 个子节点，并且节点 A 处的倒推值等于−1，将此下界叫作 MAX 节点的 α 值，即 $\alpha \geqslant -1$。现在轮到节点 B，产生它的第一个后继节点 C，节点

C 的静态估值为−1，可知节点 B 处的倒推值≤−1，此为上界 MIN 节点的 β 值，即节点 B 处 $\beta \leqslant -1$，这样节点 B 最终的倒推值可能小于−1，但绝不可能大于−1，因此，节点 B 的其他后继节点的静态估值不必计算，自然不必再生成，反正节点 B 决不会比节点 A 好，所以通过倒推值的比较，就可以减少搜索的工作量。在图 3-24 中，MIN 节点 B 的 β 值小于或等于节点 B 的先辈 MAX 节点 S 的 α 值，则节点 B 的其他后继节点可以不必再生成。

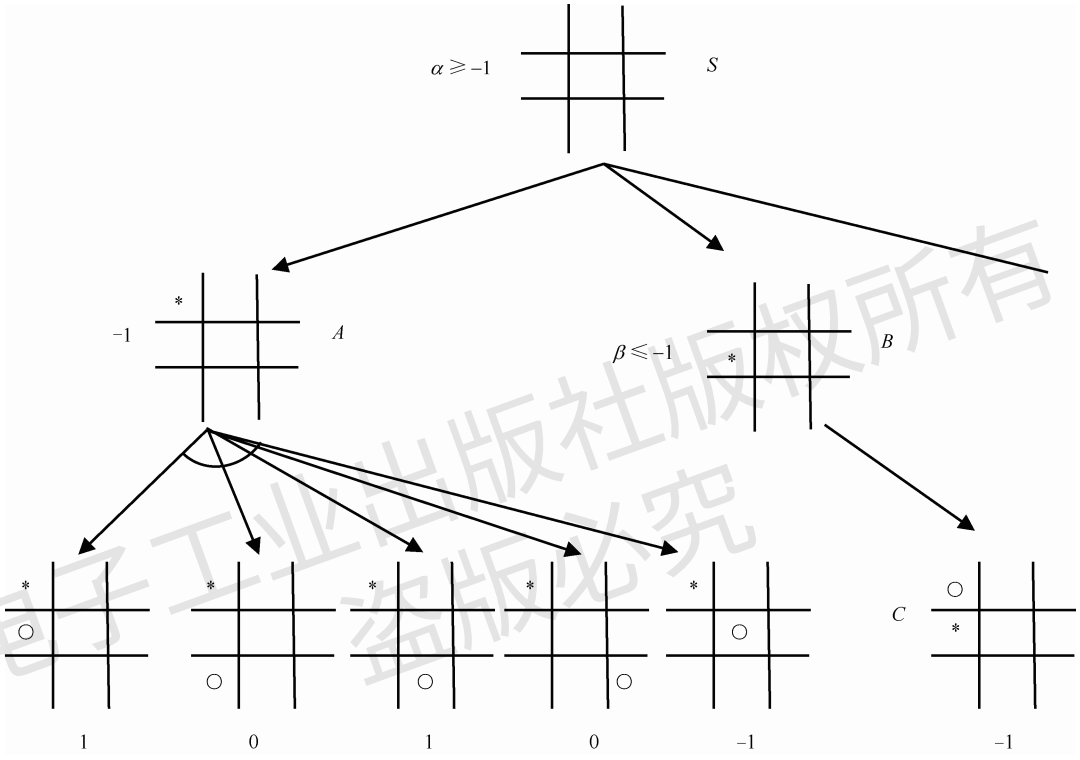


图 3-24　一字棋博弈的 α-β 过程

图 3-24 表示了 β 值小于或等于父节点的 α 值的情况，实际上，当某个 MIN 节点的 β 值不大于它先辈的 MAX 节点（不一定是父节点）的 α 值时，MIN 节点就可以终止向下搜索。同样，当某个节点的 α 值大于或等于它的先辈 MIN 节点的 β 值时，该 MAX 节点就可以终止向下搜索。

通过上面的讨论可以看出，α-β 过程首先使搜索树的某一部分达到最大深度，这时计算出某些 MAX 节点的 α 值，或者某些 MIN 节点的 β 值。随着搜索的继续，不断修改个别节点的 α 或 β 值。对任一节点，当其某一后继节点的最终值给定时，就可以确定该节点的 α 或 β 值。当该节点的其他后继节点的最终值给定时，就可以对该节点的 α 或 β 值进行修正。

注意对 α 、β 值修正有如下规律。

（1）MAX 节点的 α 值永不减小。

（2）MIN 节点的 β 值永不增大。

因此可以利用上述规律进行剪枝，一般来说可把停止对某个节点进行搜索，即剪枝的规则表述如下。

（1）若任何 MIN 节点的 β 值小于或等于任何它的先辈 MAX 节点的 α 值，则可停止该 MIN 节点以下的搜索，然后这个 MIN 节点的最终倒推值即它已得到的 β 值。该值与真正的极大极小过程搜索结果的倒推值可能不相同，但是对开始节点而言，倒推值是相同的，使用它选择的走步也是相同的。

（2）若任何 MAX 节点的 α 值大于或等于它的先辈 MIN 节点的 β 值，则可以停止该 MAX 节点以下的搜索，然后这个 MAX 节点处的倒推值即它已得到的 α 值。

当满足规则（1）而减少了搜索时，称进行了 α 剪枝；当满足规则（2）而减少了搜索时，称进行了 β 剪枝。保存 α 、β 值，并且当可能的时候就进行剪枝的整个过程通常被称为 α-β 过程，当初始节点的全体后继节点的最终倒推值全都给出时，上述过程便结束。在搜索深度相同的条件下，采用这个过程所获得的走步跟简单的极大极小过程的结果是相同的，区别只在于 α-β 过程通常只用少得多的搜索便可以找到一个理想的走步。

考察图 3-25 所示的博弈树。各端节点的估值如图所示，其中 G 尚未计算估值。由 D 与 E 的估值得到 B 的倒推值为 3，这表示 A 的倒推值最小为 3。另外，由 F 的估值得知 C 的倒推值最大为 2，因此 A 的倒推值为 3。这里，虽然没有计算 G 的估值，仍然不影响对上层节点倒推的计算，这表示这个分支可以从博弈树中减去。

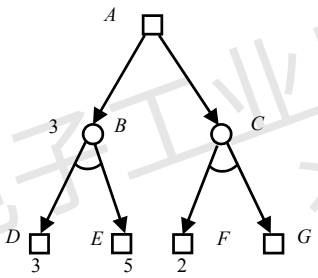


图 3-25　α-β 剪枝示例

α-β 过程的搜索效率与最先生成的节点的 α 、β 值和最终倒推值之间的近似程度有关。初始节点最终倒推值将等于某个叶节点的静态估值。如果在深度优先搜索的过程中，第一次就碰到了这个节点，则剪枝数最大，搜索效率最高。

假设一棵树的深度为 d，且每个非叶节点的分支系数为 b。对于最佳情况，即 MIN 节点先扩展出最小估值的后继节点，MAX 节点先扩展出最大估值的后继节点。这种情况可使得修剪的枝数最大。设叶节点的最小个数为 N_d ，则有

$$N_d=\begin{cases}2b^{\frac{d}{2}}-1, & d\text{为偶数}\\ b^{\frac{d+1}{2}}+b^{\frac{d-1}{2}}-1, & d\text{为奇数}\end{cases} \tag{3-10}$$

这说明，在最佳情况下，α-β 过程搜索生成深度为 d 的叶节点数目相当于极大极小过程所生成的深度为 $d/2$ 的博弈树的节点数。也就是说，为了得到最佳步数，α-β 过程只需要检测 $O(b^{d/2})$，而不是极大极小过程的 $O(b^d)$。这样，有效的分支系数是 $\sqrt{b}$，而不是 b。假设国际象棋可以有 35 种走步的选择，则现在是 6 种。从另一个角度看，在相同的代价下，α-β 过程向前看的走步数是极大极小过程向前看的走步数的两倍。

3.5　遗传算法

达尔文的进化论和孟德尔的遗传定律是人类科学史上的重要学说。在人工智能领域也有学者根据这两个学说抽象出了基于“进化”观点的学习理论，即进化计算。进化计算是一种模拟生物进化、自然选择过程与机制求解问题的自组织、自适应人工智能技术。遗传算法就是进化计算的典型。

3.5.1　基本过程


微课视频

遗传算法的核心思想认为，生物进化过程是从简单到复杂、从低级到高级的过程，其本身是一个自然的、并行的、稳健的优化过程，优化的目标是对环境具有自适应性。生物种群通过“优胜劣汰”及遗传变异来达到进化（优化）的目的。遗传算法用模拟生物和人类进化的方法来求解复杂问题。它从初始种群出发，采用“优胜劣汰，适者生存”的自然法则选择个体，并通过杂交、变异来产生新一代种群，如此逐代进化，直到满足目标为止。

遗传算法涉及的基本概念主要有以下 5 个。

（1）种群，是指用遗传算法求解问题时，初始给定的多个解的集合，是问题解空间的一个子集。遗传算法的求解过程是从这个子集开始的。

（2）个体，是指种群中的单个元素，通常由一个用于描述其基本遗传结构的数据结构来表示。例如，可以用 0 和 1 组成的串来表示个体。

（3）染色体，是指对个体进行编码后所得到的编码串。染色体中的每一个位称为基因，染色体上由若干基因构成的一个有效信息段称为基因组。

（4）适应度函数，是一种用来对种群中每个个体的环境适应性进行度量的函数。其函数值决定着染色体的优劣程度，是遗传算法实现优胜劣汰的主要依据。

（5）遗传操作，是指作用于种群而产生新的种群的操作。标准的遗传操作包括选择（或复制）、交叉（或重组）、变异三种基本形式。

遗传算法在实际操作中，需要确定一些控制参数，常见的控制参数主要如下。

（1）串长，编码字符串所含字符（或者数值）的个数，为常数。

（2）种群容量，每一代种群内个体的总数目，即种群内所含编码字符串的数目。一般情况下，每一代种群容量不变。

（3）交叉概率，就是种群中一个个体要被实施交叉算子的概率。

（4）变异概率，又称突变率，就是一个个体要被实施变异算子的概率，即一个个体发生基因突变的概率。基因突变的概率很小，一般情况下该值远远小于 0.1。

（5）进化代数。遗传算法完成一次遗传操作形成一代新种群。进化代数就是遗传算法生成新种群的次数。进化代数常用于控制遗传算法的终止。

遗传算法可形式化地描述为

$$\mathrm{GA}=\left(P(0),N,L,s,g,P,f,T\right) \tag{3-11}$$

式中，$P(0)=\{P_1(0),P_2(0),\cdots,P_n(0)\}$ 表示初始种群；N 表示种群规模；L 表示编码串的长度；s 表示选择策略；g 表示遗传算子，包括选择算子 Q_r、交叉算子 Q_c 和变异算子 Q_m；P 表示遗传算子的操作概率，包括选择概率 P_r、交叉概率 P_c 和变异概率 P_m；f 是适应度函数；T 是终止标准。

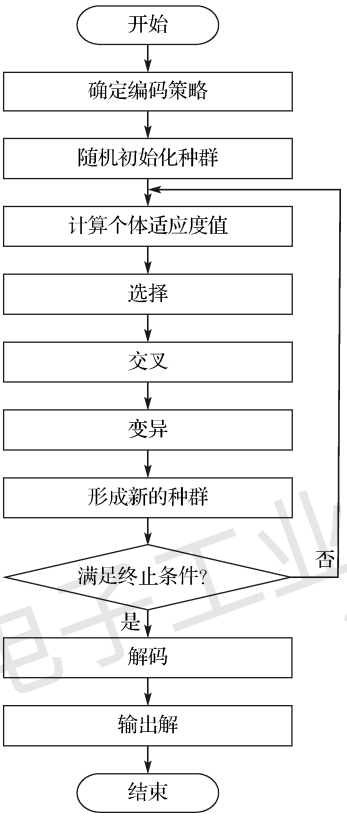


图 3-26　遗传算法的流程图

遗传算法主要由染色体编码、初始种群设定、适应度函数设定、遗传操作设计等几个部分组成，其流程图如图 3-26 所示，其主要内容和基本步骤可描述如下。

（1）确定编码策略。将问题搜索空间中每个可能的点用相应的编码策略表示出来，即形成染色体。

（2）定义遗传策略。定义种群规模 N，确定交叉、变异、选择方法，以及确定选择概率 P_r、交叉概率 P_c、变异概率 P_m 等遗传参数。

（3）令 $t=0$，随机选择 N 个染色体初始化种群 $P(0)$。

（4）定义适应度函数 $f(f>0)$。

（5）计算 $P(t)$ 中每个染色体的适应度值。

（6）$t=t+1$。

（7）运用选择算子，从 $P(t-1)$ 中得到 $P(t)$。

（8）$P(t)$ 中的每个染色体，按概率 P_c 参与交叉。

（9）染色体中的基因，以概率 P_m 参与变异运算。

（10）判断群体性能是否满足预先设定的终止条件，若不满足，则返回第（5）步。

在该算法中，编码是指把实际问题的结构变换为遗传算法的染色体结构。选择是指按照选择概率和每个个体的适应度值，从当前种群中选出若干个体。交叉是指按照交叉概率和交叉策略把两个染色体的部分基因进行交配重组，产生出新的个体。变异是指按照变异概率和变异策略使染色体中的某些基因产生变化。例如，在二进制编码方式下，变异操作只是简单地将基因的二进制数取反，即将“0”变为“1”，将“1”变为“0”。

遗传算法具有如下特点。

（1）遗传算法从问题的解集中开始搜索，是群体搜索，而不是从单个解开始。这是遗传算法与传统优化算法的极大区别。传统优化算法从单个初始值迭代求最优解，容易误入局部最优解；遗传算法从串集开始搜索，覆盖面大，利于全局择优。

（2）遗传算法求解时使用特定问题的信息极少，容易形成通用算法程序。遗传算法使用适应度这一信息进行搜索，并不需要目标函数的导数等与问题直接相关的信息。遗传算法只需适应度和串编码等通用信息，故可处理很多传统解析优化算法无法解决的问题。

（3）遗传算法有极强的容错能力。遗传算法的初始串集本身带有大量与最优解相差甚远的信息，通过选择、交叉和变异操作能迅速排除与最优解相差极大的串。这是一个强烈的滤波过程，并且是一个并行滤波机制。故而，遗传算法有很高的容错能力。

（4）遗传算法中的选择、交叉和变异都是随机操作，执行概率转移准则，而不是确定的精确规则。

（5）遗传算法具有隐含的并行性。遗传算法在种群上进行选择、交叉和变异等遗传操作，所以在每一代种群上都相当于同时搜索多个解。传统的解析法优化算法每一次迭代只能沿着一个梯度方向进行搜索，而遗传算法实际上同时搜索在多方向上的可能解。

3.5.2　遗传编码

编码机制是遗传算法的基础，运用遗传算法解决问题时，首先要对待解决问题的模型结构和参数进行编码，一般用字符串表示。对于优化问题，一个串对应一个可能的解。对于分类问题，一个串可解释为一个规则，即串的前半部分为输入，后半部分为输出或结论。常用的遗传编码有二进制编码、格雷编码、实数编码和符号编码等。针对特定问题还可以混合应用多种基本编码。

3.5.2.1　二进制编码

二进制编码将原问题的结构变换为染色体的位串结构。在二进制编码中，首先确定二进制字符串的长度 L，该长度与变量的定义域和所求问题的计算精度有关。设某一参数的取值范围是 $[U_{\min},U_{\max}]$，则二进制编码的精度为

$$\delta=\frac{U_{\max}-U_{\min}}{2^{l-1}} \tag{3-12}$$

假设某一个体的编码是 $x=[b_{L-1}\cdots b_1b_0]$，则其对应的解码公式为

$$x=U_{\min}+\left(\sum_{i=0}^{L-1}b_i2^i\right)\frac{U_{\max}-U_{\min}}{2^{l-1}} \tag{3-13}$$

例 3-8　假设变量 x 的定义域为[5,10]，要求的计算精度为10^{-5}，则需要将[5,10]至少分为 600000 个等长小区间，每个小区间用一个二进制编码串表示。于是，串长至少等于 20，原因是

$$524288=2^{19}<600000<2^{20}=1048576$$

这样，对应区间[5,10]内满足精度要求的每个值 x，都可用一个 20 位的二进制编码串 $x=[b_{19}b_{18}\cdots b_1b_0]$ 来表示。其对应的十进制数为

$$x'=\sum_{i=0}^{19}b_i\times2^i$$

对应的变量 x 的值为

$$x=5+x'\times\frac{6}{2^{20}-1}=5+\left(\sum_{i=0}^{20}b_i\times2^i\right)\times\frac{6}{2^{20}-1}$$

二进制编码的主要优点如下。

（1）自然且易于实现。二进制编码类似生物染色体的组成，其算法便于用生物遗传理论来解释，且遗传操作容易实现。

（2）能够处理的模式数目最多。模式是指能够对染色体之间的相似性进行解释的模板。这种模板是通过引入通配符“*”来实现的。通配符“*”可以认为是 1 或是 0。例如，模式

“*1*”描述了由 4 个染色体组成的染色体集合{010，011，110，111}。从理论上说，采用二进制编码，算法能处理的模式最多。

二进制编码的主要缺点如下。

（1）存在汉明悬崖。在二进制编码中，相邻二进制数的编码可能具有较大的汉明距离。例如，7 和 8 的二进制数分别为 0111 和 1000，当算法将编码从 0111 改进到 1000 时必须改变所有的位。这种较大的汉明距离无疑会降低遗传算法的搜索效率。

（2）缺乏串长的微调功能。采用二进制编码，需要先根据求解精度确定串长，串长被确定后，其长度在算法执行过程中不能改变。实际上，在算法开始阶段往往不需要太高的精度，或者说不需要太大的串长。串长太大会使算法效率下降，而要提高算法效率，就需要缩小串长，但缩小串长又会导致最优解的精度下降。这是一对矛盾，为解决这对矛盾，人们又提出了一些其他编码方法。

3.5.2.2 格雷编码

格雷编码是对二进制编码进行变换后所得到的一种编码。这种编码要求两个连续整数的编码之间只能有一个码位不同，其余码位都是完全相同的，其编码精度与同长度的二进制编码精度一样，其基本原理如下。

设有二进制编码串 $b_1,b_2,\cdots,b_n$，对应的格雷编码串为 $a_1,a_2,\cdots,a_n$，则从二进制编码到格雷编码的变换为

$$a_i=\begin{cases}b_1, & i=1\\ b_{i-1}\oplus b_i, & i>1\end{cases} \tag{3-14}$$

式中，$\oplus$ 表示模 2 加法。而从一个格雷编码串到二进制编码串的变换为

$$b_i=\sum_{i=1}^{i}a_j \pmod 2 \tag{3-15}$$

例 3-9 十进制数 7 和 8 的二进制编码分别为 0111 和 1000，而其格雷编码分别为 0100 和 1100。

格雷编码有效地解决了二进制编码存在的汉明悬崖问题。对于格雷编码，任意两个整数之差是这两个整数所对应格雷编码间的汉明距离。这也是遗传算法中使用格雷编码进行个体编码的主要原因。二进制编码单个基因座的变异可能带来巨大的差异，如从 127 变到 255。而格雷编码串之间的一位差异，对应的参数也只是微小的差别。这样就增强了遗传算法的局部搜索能力，便于对连续函数进行局部空间搜索。

3.5.2.3 实数编码

实数编码是指将每个个体的染色体都用某一范围的一个实数（浮点数）来表示，其编码长度等于该问题变量的个数。这种编码方法将问题的解空间映射到实数空间上，然后在实数空间上进行遗传操作。由于实数编码使用的是变量的真实值，因此这种编码方法也称为真值编码方法。实数编码适用于多维、高精度的连续函数优化问题。

例如，若某一个优化问题含有 5 个变量 $x_i=(i=1,2,\cdots,5)$，每个变量都具有其对应的上下限，则 x：[5.80 6.90 3.50 3.80 5.00]就表示了一个个体的基因型。

在实数编码方法中，必须保证基因值在给定的区间限制范围内。遗传算法所使用的交叉、变异的遗传算子也必须保证其运算结果所产生新个体的基因值也在该区域限制范围内。

实数编码的主要优点如下。

（1）适合在遗传算法中表示范围较大的数。

（2）适合精度要求较高的遗传算法。

（3）便于较大空间的遗传搜索。

（4）改善了遗传算法的计算复杂性，提高了运算效率。

（5）便于遗传算法与经典优化方法的混合使用。

（6）便于设计针对问题的专门知识的知识型遗传算子。

（7）便于处理复杂的决策变量约束条件。

3.5.2.4　符号编码

符号编码方法是指个体编码串的基因值取自一个无数值含义只有代码含义的符号集。这个符号集可以是一个字母，如{A，B，C，D，…}；也可以是一个数字序号，如{1，2，3，4，…}；还可以是一个代码，如{C_1，C_2，C_3，…}等。

对于使用符号编码的遗传算法，需要认真设计交叉、变异等遗传运算操作方法，以满足问题的各种约束要求。

符号编码的主要优点如下。

（1）符合有意义积木块编码原则。

（2）便于遗传算法与相关近似算法之间的混合使用。

（3）便于在遗传算法中利用所求解问题的专门知识。

3.5.3　适应度函数


微课视频

适应度函数是一种对个体的适应性进行度量的函数。通常，个体的适应度值越大，它被遗传到下一代种群中的概率就越大。

3.5.3.1　常用的适应度函数

在遗传算法中，有许多计算适应度值的方法，其中最常用的适应度函数有以下两种。

1. 原始适应度函数

它是指直接将待求解问题的目标函数 $f(x)$ 定义为遗传算法的适应度函数。例如，在求解极值问题

$$\max_{x\in[a,b]} f(x) \tag{3-16}$$

时，$f(x)$ 为 x 的原始适应度函数。

采用原始适应度函数的优点是，能够直接反映出待求解问题的最初求解目标；其缺点是，有可能出现适应度值为负的情况。

2. 标准适应度函数

遗传算法中一般要求适应度函数非负，并且适应度值越大越好。这就需要对原始适应

度函数进行某种变换，将其转换为标准的度量方式，以满足进化操作的要求，这样得到的适应度函数被称为标准适应度函数 $f_{\text{normal}}(x)$。

对于极小化问题，其标准适应度函数可定义为

$$f_{\text{normal}}(x)=\begin{cases} f_{\max}(x)-f(x), & f(x)<f_{\max}(x) \\ 0, & \text{否则} \end{cases} \tag{3-17}$$

式中，$f_{\max}(x)$ 是原始适应度函数 $f(x)$ 的一个上界。如果 $f_{\max}(x)$ 未知，则可用当前代或到目前为止各演化代中的 $f(x)$ 的最大值来代替。可见，$f_{\max}(x)$ 是会随着进化代数的增加而不断变化的。

对于极大化问题，其标准适应度函数可定义为

$$f_{\text{normal}}(x)=\begin{cases} f(x)-f_{\min}(x), & f(x)>f_{\min}(x) \\ 0, & \text{否则} \end{cases} \tag{3-18}$$

式中，$f_{\min}(x)$ 是原始适应度函数 $f(x)$ 的一个下界。如果 $f_{\min}(x)$ 未知，则可用当前代或到目前为止各演化代中的 $f(x)$ 的最小值来代替。

3.5.3.2 适应度函数的加速变换

在某些情况下，适应度函数在极值附近的变化可能非常小，以至于不同个体的适应度值非常接近，难以区分出哪个染色体更占优势。对此，最好能定义新的适应度函数，使该适应度函数既与问题的目标函数具有相同的变化趋势，也有更快的变化速度。适应度函数的加速变换有两种基本方法：线性加速、非线性加速。下面重点讨论线性加速问题。

线性加速适应度函数的定义如下：

$$f'(x)=af(x)+\beta \tag{3-19}$$

式中，$f(x)$ 是加速转换前的适应度函数；$f'(x)$ 是加速转换后的适应度函数：α和β 是转换系数。α和β 的选择应满足如下条件。

（1）变换后得到的新的适应度函数的平均值要等于原适应度函数的平均值。这样可以保证父代种群中的适应度值接近于平均适应度值的个体，能够有相当数量被遗传到下一代种群中，即

$$a\times\frac{\sum_{i=1}^{n}f(x_i)}{n}+\beta=\frac{\sum_{i=1}^{n}f(x_i)}{n} \tag{3-20}$$

式中，$x_i(i=1,2,\cdots,n)$ 为当前代中的染色体。

（2）变换后得到的新的种群个体所具有的最大适应度值要等于其平均适应度值的指定倍数，即

$$a\times\max_{1\leqslant i\leqslant n}\{f(x_i)\}+\beta=M\times\frac{\sum_{i=1}^{n}f(x_i)}{n} \tag{3-21}$$

式中，$x_i(i=1,2,\cdots,n)$ 为当前代中的染色体；M 是指将当前代中的最大适应度值放大为其平均值的 M 倍。这样，通过选择适当的 M 值，就可以拉开不同染色体间适应度值的差距。

关于α和β的值，可通过求解由式（3-20）和式（3-21）所组成的联立方程组而得到。

除采用线性加速变换办法外，也可采用非线性加速变换方法。

幂函数变换方法：

$$f'(x)=f(x)^k \tag{3-22}$$

指数函数变换方法：

$$f'(x)=\exp\left(-\beta f(x)\right) \tag{3-23}$$

3.5.4　遗传操作

遗传算法中的基本遗传操作包括选择、交叉和变异三种，每种操作又包括多种方法，下面分别进行介绍。

3.5.4.1　选择操作

选择操作是指根据适者生存原则从种群中选择出生命力强、较适应环境的个体。这些被选中的个体用于繁殖下一代，产生新种群，故这一操作也称为繁殖。由于在选择用于繁殖下一代的个体时，根据个体对环境的适应度而决定其是否繁殖，所以还称其为非均匀繁殖。

选择操作以适应度为选择原则，体现出优胜劣汰的效果。具体的选择方法很多，都遵从两个原则：①适应度值较高的个体繁殖下一代的概率较高；②适应度值较低的个体繁殖下一代的概率较低，甚至会被淘汰。

常见的选择操作有如下几种。

1. 比例选择

比例选择的基本思想是，每个个体被选中的概率与其适应度值的大小成正比。由于随机操作的原因，这种选择方法的选择误差比较大。常用的比例选择法包括轮盘赌选择法和繁殖池选择法等。

轮盘赌选择法又称为转盘赌选择法或轮盘选择法，是比例选择法中最常用的一种方法。该方法的基本思想是，个体被选中的概率取决于该个体的相对适应度。相对适应度定义为

$$P(x_i)=\frac{f(x_i)}{\sum_{j=1}^{N} f(x_j)} \tag{3-24}$$

式中，$P(x_i)$是第i个个体x_i的相对适应度，即个体x_i被选中的概率；$f(x_i)$是个体x_i的原始适应度；$\sum_{j=1}^{N} f(x_j)$是种群的累加适应度。

轮盘赌选择法的基本思想是，根据每个个体被选中的概率$P(x_i)$，将圆盘分成N个扇区，其第i个扇区的中心角为

$$2\pi\frac{f(x_i)}{\sum_{j-1}^{N} f(x_j)}=2\pi P(x_i) \tag{3-25}$$

再设立一个固定指针。当进行选择时，可以假想转动圆盘，若圆盘静止时指针指向第

i 个扇区，则选择个体 x_i。其物理意义如图 3-27 所示。

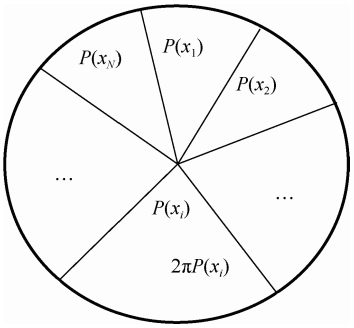


图 3-27　轮盘赌选择法的物理意义

从统计角度看，个体的适应度值越大，其对应的扇区的面积越大，被选中的可能性越大。从而，其基因被遗传到下一代的可能性也就越大。反之，适应度值越小的个体，被选中的可能性就越小，但仍有被选中的可能。这种方法有点类似发放奖品时使用的轮盘，并带有某种赌博的意思，因此被称为轮盘赌选择。

繁殖池选择法也是比例选择法中常用的一种方法，其基本思想是，首先计算种群中每个个体的繁殖数目 N_i，并分别把每个个体复制成 N_i 个个体；其次将这些复制后的个体组成一个临时种群，即形成一个繁殖池；再次从繁殖池中成对地随机抽取个体进行交叉操作，并用新产生的个体取代当前个体；最后形成下一代个体种群。

种群中第 i 个个体的繁殖数目 N_i 可按式（3-26）进行计算：

$$N_i = \text{round}\left(\text{rel}_i \times N\right) \tag{3-26}$$

式中，$\text{round}\left(x\right)$ 表示与 x 距离最小的整数；N 表示种群规模；rel_i 表示种群中第 i 个个体的相对适应度，其计算公式为

$$\text{rel}_i = \frac{f\left(x_i\right)}{\sum_{j=1}^{N} f\left(x_j\right)} \tag{3-27}$$

式中，$f\left(x_i\right)$ 是种群中第 i 个个体的适应度值。

可以看出，个体的适应度值越大，其相对适应度值越大和繁殖数目越多，即它在繁殖池中被选中的机会就越大。而那些 $N_i = 0$ 的个体肯定会被淘汰。

2. 排序选择

排序选择的基本思想是，首先对种群内的所有个体，按其相对适应度值的大小进行排序；然后根据每个个体的排列顺序，为其分配相应的选择概率；最后基于这些选择概率，采用比例选择法（如轮盘赌选择法）产生下一代种群。

这种方法的主要优点是，消除了个体适应度值因差别很大所产生的影响，使每个个体的选择概率仅与其在种群中的排序有关，而与其适应度值无直接关系。其主要缺点：一是忽略了适应度值之间的实际差别，使得个体的遗传信息未能得到充分利用；二是选择概率和序号的关系必须事先确定。

3. 竞技选择

竞技选择也称为锦标赛选择，其基本思想：首先在种群中随机选择 k 个（允许重复）个体进行锦标式比较，适应度值大的个体将胜出，并被作为下一代种群中的个体；重复以上过程，直到下一代种群中的个体数目达到种群规模为止。参数 k 被称为竞赛规模，通常取 $k = 2$。

这种方法实际上是将局部竞争引入到了选择过程中，既能使那些好的个体有较多的繁殖机会，也可避免某个个体因其适应度值过大而在下一代繁殖较多的情况。

4. 无回放随机选择

这种选择方法也称作期望值选择方法。它的基本思想是，根据每个个体在下一代群体中的生存期望值来进行随机选择。其具体操作过程如下所述。

计算群体中每个个体在下一代群体中的生存期望数目 n_i：

$$n_i = N\frac{f(x_i)}{\sum_{j=1}^{N} f(x_j)} \tag{3-28}$$

若某一个个体被选中参与交叉运算，则它在下一代中的生存期望值减小 0.5；若某一个个体未被选中，则它在下一代中的生存期望值减小 1.0。随着选择过程的进行，若某一个个体的生存期望值小于 0，则该个体不再有机会被选中。

这种选择操作方法能够降低一些误差，但操作不太方便。

3.5.4.2　交叉操作

交叉操作就是指在选中用于繁殖下一代的个体（染色体）中，对两个不同染色体相同位置上的基因进行交换，从而产生新的染色体，所以交叉操作又称为重组操作。当许多染色体相同或后代的染色体与上一代没有多大差别时，可通过染色体重组来产生新一代染色体。交叉操作分为两个步骤：首先进行随机配对，然后执行交叉操作。配对就是指从被选中用于繁殖下一代的个体中，随机地选取两个个体组成一对。交叉操作就是按照一定概率在某位置上交换配对编码的部分子串。这是一个随机信息交换过程，其目的在于产生新的基因组合，即产生新的个体。根据个体编码方法的不同，遗传算法中的交叉操作可分为二进制值交叉和实值交叉两种类型。

1. 二进制值交叉

二进制值交叉是指在二进制编码情况下采用的交叉操作，主要包括单点交叉、两点交叉、多点交叉和均匀交叉等方法。

1）单点交叉

单点交叉也称为简单交叉，是指先在两个父代个体的编码串中随机设定一个交叉点，然后对这两个父代个体交叉点前面或后面部分的基因进行交换，并生成子代中的两个新的个体。

假设两个父代个体的编码串分别是

$$X = x_1\ x_2 \cdots x_k\ x_{k+1} \cdots x_n$$

$$Y = y_1\ y_2\ \cdots y_k\ y_{k+1} \cdots y_n$$

随机选择第 k 位为交叉点，若采用对交叉点后面的基因进行交换的方法，即将 X 中的 $x_{k+1} \sim x_n$ 部分与 Y 中的 $y_{k+1} \sim y_n$ 部分进行交换，交换后生成的两个新个体的编码串是

$$X' = x_1\ x_2 \cdots x_k\ y_{k+1} \cdots y_n$$

$$Y' = y_1\ y_2\ \cdots y_k\ x_{k+1} \cdots x_n$$

交叉得到的新编码串不一定都能保留在下一代，可以仅保留适应度值大的那个编码串。

单点交叉的特点：若邻接基因座之间的关系能提供较好的个体性状和较大的个体适应度值，则这种单点交叉操作破坏这种个体性状和较小个体适应度值的可能性最小。

例 3-10 设有两个父代个体的编码串 $A=001101$ 和 $B=110010$，若随机交叉点为 4，则交叉后生成的两个新个体的编码串是

$$A'=001110$$
$$B'=110001$$

2）两点交叉

两点交叉是指先在两个父代个体的编码串中随机设定两个交叉点，再按这两个交叉点进行部分基因交换，生成子代中的两个新的个体。

假设两个父代个体的编码串分别是

$$X=x_1\ x_2\cdots x_i\cdots x_j\cdots x_n$$
$$Y=y_1\ y_2\ \cdots y_i\cdots y_j\cdots y_n$$

随机设定第 i、j 位为两个交叉点（其中$i<j<n$），即将 X 中的 $x_{i+1}\sim x_j$ 部分与 Y 的 $y_{i+1}\sim y_j$ 部分进行交换，交叉后生成的两个新个体的编码串是

$$X'=x_1\ x_2\cdots x_i\ y_{i+1}\cdots y_j\ x_{j+1}\cdots x_n$$
$$Y'=y_1\ \ y_2\cdots y_i\ x_{i+1}\cdots x_j\ y_{j+1}\cdots y_n$$

例 3-11 设有两个父代个体的编码串 $A=001011$ 和 $B=110100$，若随机交叉点为 3 和 5，则交叉后生成的两个新个体的编码串是

$$A'=001101$$
$$B'=110010$$

3）多点交叉

多点交叉是指先在两个父代个体的编码串中随机生成多个交叉点，再按这些交叉点分段地进行部分基因交换，生成子代中的两个新的个体。

假设设置的交叉点个数为 m 个，则可将个体串（染色体）划分为 $m+1$ 个分段（基因组），其划分方法：当 m 为偶数时，对全部交叉点依次进行两两配对，构成 $m/2$ 个交叉段；当 m 为奇数时，对前 $m-1$ 个交叉点依次进行两两配对，构成 $(m-1)/2$ 个交叉段，而第 m 个交叉点按单点交叉方法构成一个交叉段。

为便于理解，下面以 $m=3$ 为例进行讨论。假设两个父代个体的编码串分别是

$$X=x_1\ x_2\cdots x_i\cdots x_j\cdots x_k\cdots x_n$$
$$Y=y_1\ y_2\ \cdots y_i\cdots y_j\cdots y_k\cdots y_n$$

随机设定第 i、j、k 位为三个交叉点（$i<j<k<n$），则将构成两个交叉段。其中，第一个交叉段是由前两个交叉点构成一个两点交叉段，即将 X 中的 $x_{i+1}\sim x_j$ 部分与 Y 中的 $y_{i+1}\sim y_j$ 部分进行交换；第二个交叉段是由第三个交叉点构成的一个单点交叉段，即将 X 中的 $x_{k+1}\sim x_n$ 部分与 Y 中的 $y_{k+1}\sim y_n$ 部分进行交换。交叉后生成的两个新个体的编码串是

$$X'=x_1\ x_2\cdots x_i\ y_{i+1}\cdots y_j\ x_{j+1}\cdots x_k\ y_{k+1}\cdots y_n$$
$$Y'=y_1\ \ y_2\cdots y_i\ x_{i+1}\cdots x_j\ y_{j+1}\cdots y_k\ x_{k+1}\cdots x_n$$

例 3-12　设有两个父代个体的编码串 $A=001101$ 和 $B=110010$，若随机交叉点为 1、3 和 5，则交叉后生成的两个新个体的编码串是

$$A'=010100$$
$$B'=101011$$

4）均匀交叉

均匀交叉先随机生成一个与父代个体的编码串具有相同长度，并被称为交叉模板（或交叉掩码）的二进制串，再利用该模板对两个父代个体的编码串进行交叉，即将模板中1对应的位进行交换，而 0 对应的位不交换，依次生成子代中的两个新的个体。事实上，这种方法对父代个体的编码串中的每一位都是以相同的概率随机进行交叉的。

例 3-13　设有两个父代个体的编码串 $A=001101$和$B=110010$，若随机生成的模板 $T=010011$，则交叉后生成的两个新个体的编码串是 $A'=011110$ 和 $B'=100001$，即

$$T=010011$$
$$A'=011110$$
$$B'=100001$$

2. 实值交叉

实值交叉是在实数编码情况下所采用的交叉操作，包括离散交叉和算术交叉等。

1）离散交叉

离散交叉又可分为部分离散交叉和整体离散交叉。部分离散交叉是指先在两个父代个体的编码向量中随机选择一部分分量，然后对这部分分量进行交换，生成子代中的两个新的个体。整体离散交叉则是指对两个父代个体的编码向量中的所有分量，都以1/2的概率进行交换，从而生成子代中的两个新的个体。

对于部分离散交叉，假设两个父代个体的 n 维实向量分别是 $\boldsymbol{X}=[x_1\ x_2\cdots x_i\cdots x_k\cdots x_n]$ 和 $\boldsymbol{Y}=[y_1\ y_2\ \cdots y_i\cdots y_k\cdots y_n]$，若随机选择第 k 个分量以后的所有分量进行交换，则生成的两个新个体的向量是

$$\boldsymbol{X}'=[x_1\ x_2\cdots x_k\ y_{k+1}\cdots y_n]$$
$$\boldsymbol{Y}'=[y_1\ y_2\ \cdots y_k\ \ x_{k+1}\cdots x_n]$$

例 3-14　设有两个父代个体的向量 $\boldsymbol{A}$=[20 16 19 32 18 26]和 $\boldsymbol{B}$=[36 25 38 12 21 30]，若随机选择第 3 个分量以后的所有分量进行交换，则交换后的两个新个体的向量是

$$\boldsymbol{A}'=\ [20\ 16\ 19\ 12\ 21\ 30]$$
$$\boldsymbol{B}'=\ [36\ 25\ 38\ 32\ 18\ 26]$$

2）算数交叉

算数交叉是指由两个个体线性组合而产生出两个新的个体。

假设在两个个体 X_A^t、X_B^t 之间进行算数交叉，则交叉后所产生的两个新个体是

$$\begin{cases}X_A^{t+1}=\alpha X_B^t+(1-\alpha)X_A^t\\X_B^{t+1}=\alpha X_A^t+(1-\alpha)X_B^t\end{cases}$$

式中，α 为一个参数，它可以是一个常数，此时所进行的交叉运算称为均匀算数交叉；它也可以是一个由进化代数所决定的变量，此时所进行的交叉运算称为非均匀算数交叉。

3.5.4.3 变异操作


微课视频

变异也称为突变，是指对选中个体的染色体中的某些基因执行异向转化，以形成新的个体。变异也是生物遗传和自然进化中的一种基本现象，可增强种群的多样性。遗传算法中的变异操作增加了算法的局部随机搜索能力，从而可以维持种群的多样性。根据个体编码方式的不同，变异操作可分为二进制值变异和实值变异两种类型。

1. 二进制值变异

当个体的染色体用二进制编码表示时，其变异操作应采用二进制值变异方法。该变异方法是，先随机地产生一个变异位，然后将该变异位上的基因值由“0”变为“1”或由“1”变为“0”，产生一个新的个体。

例 3-15 设变异前个体的编码串为 $A=001101$，若随机产生的变异位置是 2，则该个体的第 2 位将由“0”变为“1”，变异后的新个体的编码串是 $A'=011101$。

2. 实值变异

当个体的染色体用实数编码表示时，其变异操作应采用实值变异方法。该方法是，用另一个在规定范围内的随机实数去替换原变异位上的基因值，产生一个新的个体。最常用的实值变异操作有基于位置的变异和基于次序的变异等。

基于位置的变异方法是，先随机地产生两个变异位置，然后将第二个变异位置上的基因移动到第一个变异位置的前面。

例 3-16 设选中的个体向量 $\boldsymbol{C}=[20\ 16\ 19\ 12\ 21\ 30]$，若随机产生的两个变异位置分别是 2 和 4，则变异后的新的个体向量是

$$\boldsymbol{C}'=[20\ 12\ 16\ 19\ 21\ 30]$$

基于次序的变异方法是，先随机地产生两个变异位置，然后交换这两个变异位置上的基因。

例 3-17 设选中的个体向量 $\boldsymbol{D}=[20\ 12\ 16\ 19\ 21\ 30]$，若随机产生的两个变异位置分别是 2 和 4，则变异后的新的个体向量是

$$\boldsymbol{D}'=[20\ 19\ 16\ 12\ 21\ 30]$$

例 3-18 用遗传算法求函数 $f(x)=x^2$ 的最大值。式中，x 为 $[0,31]$ 上的整数。

解：

（1）编码。

由于 x 是区间 $[0,31]$ 上的整数，用 5 位二进制数即可全部表示，因此可采用二进制编码方法，其编码串长度为 5。例如，用二进制编码串 00000 来表示 $x=0$，用 11111 表示 x=31 等。其中的 0 和 1 为基因值。

（2）生成初始种群。

假设种群规模 $N=4$，然后从全部个体中随机抽取 4 个个体组成初始种群。随机生成的初始种群（即第 0 代种群）表示为

$$S_{01}=01101$$
$$S_{02}=11001$$

$$S_{03}=01000$$
$$S_{04}=10010$$

（3）计算适应度值。

这里采用原始适应度函数计算其适应度值，将二进制编码转换成十进制整数，然后取其二次方作为该个体的适应度值，即

$$f(S)=f(x)$$

式中的二进制编码串 S 对应着变量 x 的值。根据此函数，初始种群情况如表 3-1 所示。可以看出，在 4 个个体中，S_{02} 的适应度值最大，是当前最佳个体。

表 3-1　初始种群情况

编　　号	个体编码串（染色体）	x	适 应 度 值	百分比/%	累计百分比/%	选 中 次 数
S_{01}	0 1 1 0 1	13	169	14.30	14.30	1
S_{02}	1 1 0 0 1	25	625	52.88	67.18	2
S_{03}	0 1 0 0 0	8	64	5.41	72.59	0
S_{04}	1 0 0 1 0	18	324	27.41	100	1

（4）选择操作。

采用简单的轮盘赌选择法选择个体，且依次生成的 4 个随机数（相当于轮盘上指针所指的数）为 0.85、0.32、0.12 和 0.46，经选择后得到的新的种群为

$$S_{01}=10010$$
$$S_{02}=11001$$
$$S_{03}=01101$$
$$S_{04}=11001$$

其中，染色体 11001 在种群中出现了两次，而原染色体 01000 则因适应度值太小而被淘汰。

（5）交叉。

令交叉概率 P_c 为 50%，即种群中只有 1/2 的染色体参与交叉。若规定种群中的染色体按顺序两两配对交叉，且有 S_{01} 与 S_{02} 交叉、S_{03} 与 S_{04} 不交叉，则初始种群的交叉情况如表 3-2 所示。

表 3-2　初始种群的交叉情况

编　　号	个体编码串（染色体）	交 叉 对 象	交　叉　位	子　　代	适 应 度 值
S_{01}	1 0 0 1 0	S_{02}	3	1 0 0 0 1	289
S_{02}	1 1 0 0 1	S_{01}	3	1 1 0 1 0	676
S_{03}	0 1 1 0 1	S_{04}	N	0 1 1 0 1	169
S_{04}	1 1 0 0 1	S_{03}	N	1 1 0 0 1	625

经交叉后得到的新的种群为

$$S_{01}=10001$$
$$S_{02}=11010$$
$$S_{03}=01101$$
$$S_{04}=11001$$

（6）变异。

变异概率 P_m 一般都很小，此处令变异概率 P_m 为 0.01，即平均每 100 位中有 1 位突变。本例题的群体只包含 4 个 5 位的字符串共 20 位。平均每遗传一代只有 0.2 位产生突变，每遗传 5 代才有 1 位发生突变。假设本次循环中没有发生变异，则变异前的种群为进化后所得到的第 1 代种群，即

$$S_{11}=10001$$
$$S_{12}=11010$$
$$S_{13}=01101$$
$$S_{14}=11001$$

对第 1 代种群重复上述第（4）～（6）步的操作。

对于第 1 代种群，其选择情况如表 3-3 所示。

表 3-3　第 1 代种群的选择情况

编　号	个体编码串（染色体）	x	适 应 值	百分比/%	累计百分比/%	选 中 次 数
S_{11}	10001	17	289	16.43	16.43	1
S_{12}	11010	26	676	38.43	54.86	2
S_{13}	01101	13	169	9.61	64.47	0
S_{14}	11001	25	625	35.53	100	1

其中，若假设按轮盘赌选择法依次生成的 4 个随机数为 0.14、0.51、0.24 和 0.82，则经选择后得到的新的种群为

$$S_{11}=10001$$
$$S_{12}=11010$$
$$S_{13}=11010$$
$$S_{14}=11001$$

可以看出，染色体 11010 被选择了两次，而原染色体 01101 因适应度值太小而被淘汰。

对于第 1 代种群，若交叉概率为 1，则其交叉情况如表 3-4 所示。

表 3-4　第 1 代种群的交叉情况

编　号	个体编码串（染色体）	交 叉 对 象	交　叉　位	子　代	适 应 度 值
S_{11}	10001	S_{12}	3	10010	324
S_{12}	11010	S_{11}	3	11001	625
S_{13}	11010	S_{14}	2	11001	625
S_{14}	11001	S_{13}	2	11010	676

可见，经交叉后得到的新的种群为

$$S_{11}=10010$$
$$S_{12}=11001$$
$$S_{13}=11001$$
$$S_{14}=11010$$

从这个新的种群来看，第 3 位基因均为 0，已经不可能通过交叉达到最优解。这种过早陷入局部最优解的现象称为早熟。为解决这一问题，需要采用变异操作。

对于第 1 代种群，其变异情况如表 3-5 所示。

表 3-5　第 1 代种群的变异情况

编　号	个体编码串（染色体）	是否变异	变异位	子　代	适应度值
S_{11}	10010	否		10010	324
S_{12}	11001	否		11001	625
S_{13}	11001	否		11001	625
S_{14}	11010	是	3	11110	900

它是通过对 S_{14} 的第 3 位进行变异来实现的。变异后得到的第 2 代种群为

$$S_{21}=10010$$
$$S_{22}=11001$$
$$S_{23}=11001$$
$$S_{24}=11110$$

对第 2 代种群同样重复上述第（4）～（6）步的操作。

对于第 2 代种群，其选择情况如表 3-6 所示。

表 3-6　第 2 代种群的选择情况

编　号	个体编码串（染色体）	x	适　应　值	百分比/%	累计百分比/%	选中次数
S_{21}	10010	18	324	13.10	13.10	1
S_{22}	11001	25	625	25.26	38.36	1
S_{23}	11001	25	625	25.26	63.62	1
S_{24}	11110	30	900	36.38	100	1

其中，若假设按轮盘赌选择法依次生成的 4 个随机数为 0.42、0.15、0.59 和 0.91，则经选择后得到的新的种群为

$$S_{21}=11001$$
$$S_{22}=10010$$
$$S_{23}=11001$$
$$S_{24}=11110$$

对于第 2 代种群，其交叉情况如表 3-7 所示。这时，函数的最大值已经出现，其对应

的染色体为 11111，经解码后可知，问题的最优解是 $x=31$。

表 3-7　第 2 代种群的交叉情况

编　　号	个体编码串（染色体）	交 叉 对 象	交　叉　位	子　　代	适 应 度 值
S_{21}	1 1 0 0 1	S_{22}	3	1 1 0 1 0	676
S_{22}	1 0 0 1 0	S_{21}	3	1 0 0 0 1	289
S_{23}	1 1 0 0 1	S_{24}	4	1 1 0 0 0	576
S_{24}	1 1 1 1 0	S_{23}	4	1 1 1 1 1	961

3.6　智能搜索应用案例

本节以一个元素选取实例来直观地理解遗传算法。

在一个长度为 50 的数组中，选取 10 个元素，要求这 10 个元素的和为数组内所有元素总和的 1/10 或尽可能接近。若采用穷举法解决该问题，从 50 个元素中选取 10 个元素，有 C_{50}^{10} 种选取结果，计算量非常大。下面采用遗传算法对该问题进行求解。

（1）创建随机个体，Python 参考代码如下。

```
def create_answer(number_set, n): # number_set 为数组， n 为创建的个体数目
    result = []
    for i in range (n):
        result.append(random.sample(number_set, 10))
    return result
```

（2）计算 error，用于选择交换个体。error 越大，其与真实值越接近，选取到的概率也就越大，其信息更容易保存下来（优胜劣汰）。Python 参考代码如下。

```
def error_level(new_answer, numbers_set): # new_answer 为所有个体
    error = []
    right_answer = sum(numbers_set)/10
    for item in new_answer:
        value = abs(right_answer - sum(item))
        if value == 0:
            error.append(10)# value 次小值为 0.1,1/value 为 10，这里大于或等于 10 都行
        else:
            error.append(1/value) # value 越小越好，即 error 越大
    return error

def choice_selected(old_answer, numbers_set):
    result = []
    error = error_level(old_answer,numbers_set)
```

```
        error_one = [item/sum(error) for item in error] #error 归一化，所有个体的 error 和为 1，这里 error 可视为后面选取的概率
        for i in range(1,len(error_one)):
            error_one[i] += error_one[i-1] #前面  error 求和，方便个体抽取
```

（3）交换信息，Python 参考代码如下。

```
    def choice_selected(old_answer, numbers_set):
        result = [] #  保存一次繁衍的所有子代
        error = error_level(old_answer,numbers_set)
        error_one = [item/sum(error) for item in error]
        for i in range(1,len(error_one)):
            error_one[i] += error_one[i-1]

        for i in range(len(old_answer)//2): #繁衍次数
            temp=[]
            for j in range(2):
                rand = random.uniform(0, 1)
                for k in range(len(error_one)):
                    if k == 0:
                        if rand<error_one[k]:
                            temp.append(old_answer[k])
                    else:
                        if rand>=error_one[k-1] and rand<error_one[k]:
                            temp.append(old_answer[k])
            rand = random.randint(0, 6) #  选择连续的 3 个数进行交换，一共 10 个数
            temp_1 = temp[0][:rand] + temp[1][rand:rand+3] + temp[0][rand+3:]
            temp_2 = temp[1][:rand] + temp[0][rand:rand+3] + temp[1][rand+3:]
            if len(set(temp_1)) == 10 and len(set(temp_2)) == 10: #  判断子代是否具有相同元素，若有则保存父本和母本
                result.append(temp_1)
                result.append(temp_2)
            else:
                result.append(temp[0])
                result.append(temp[1])
        return result
```

（4）信息变异，Python 代码如下。

```
    def variation(old_answer, numbers_set, pro): # pro 为产生变异的概率
        for i in range(len(old_answer)):
            rand = random.uniform(0, 1)
            if rand < pro:
                rand_num = random.randint(0, 9)
```

```
                new_numbers_set = [i for i in numbers_set if i not in old_answer] # 变异的数不能与个体其他数相同
                old_answer[i] = old_answer[i][:rand_num] + random.sample(new_numbers_set, 1) + old_answer[i][rand_num+1:]
```

（5）结果打印。在 0 到 1000 中随机选取 50 个数作为一个原始的数组，按照相应要求执行程序打印结果，Python 代码如下。

```
import random
numbers_set = random.sample(range(0,1000), 50) # 在 0 到 1000 中随机选取 50 个数作为一个原始的数组
middle_answer = create_answer(numbers_set, 100)# 随机创建 100 个个体
first_answer = middle_answer[0]
great_answer = [ ] # 用来保存每一代最好的个体
for i in range(1000):# 共繁衍 1000 代
    middle_answer = choice_selected(middle_answer, numbers_set) # 交换信息后的个体
    middle_answer = variation(middle_answer, numbers_set, 0.1) # 个体产生变异
    error = error_level(middle_answer, numbers_set) # 计算每个个体的元素和与数组和 1/10 的 error
    index = error.index(max(error)) # 保存 error 最大个体的下标（error 越大越好，后面有解释）
    great_answer.append([middle_answer[index],error[index]])# 保存这一代最好的个体和 error
great_answer.sort(key= lambda x:x[1], reverse=True)# 按照 error 从大到小排序
print("正确答案为", sum(numbers_set)/10)
print("给出的最优解为", great_answer[0][0])
print("该和为", sum(great_answer[0][0]))
print("选择系数为", great_answer[0][1])
```

元素选取结果如图 3-28 所示。

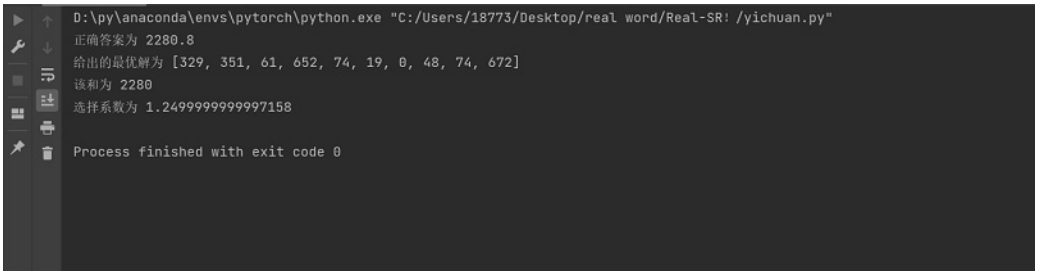


图 3-28　元素选取结果

本章小结

本章主要介绍了智能搜索策略的相关概念和基本理论，同时围绕启发式搜索方式介绍了一些常用的搜索策略，学习目标如下所述。

（1）了解并熟悉搜索的概念和相关基本知识。

（2）理解启发式搜索的含义。

（3）掌握启发式搜索、与或树搜索及博弈搜索算法。

（4）掌握遗传算法的应用及典型应用实例。

习题

一、选择题

1．关于与或图表示法的叙述，正确的是（　　）。

A 用“AND”和“OR”连接各部分的图形，用来描述各部分的因果关系

B 用“AND”和“OR”连接各部分的图形，用来描述各部分之间的不确定关系

C 用“与”节点和“或”节点组合起来的树形图，用来描述某类问题的求解过程

D 用“与”节点和“或”节点组合起来的树形图，用来描述某类问题的层次关系

2．在与或树、与或图中，把没有任何父节点的节点叫作（　　）。

A 叶节点　　B 端节点　　C 根节点　　D 起始节点

3．在启发式搜索中，通常 OPEN 表上的节点按照它们的估价函数值的（　　）顺序排列。

A 递增　　B 平均值　　C 递减　　D 最小

4．启发式搜索方法能够保证在搜索树中找到一条通向目标节点的（　　）路径（如果有路径存在时）。

A 可行　　B 最短　　C 最长　　D 解

5．下列属于遗传算法基本内容的是（　　）。

A 图像识别　　B 遗传算子　　C 语音识别　　D 神经调节

6．A*算法是一种（　　）。

A 图搜索策略　　B 有序搜索算法　　C 盲目搜索　　D 启发式搜索

二、简答题

1．什么是搜索？有哪两大类不同的搜索方法？两者的区别是什么？

2．何为估价函数？在估价函数中，$g(n)$和 $h(n)$各起什么作用？

3．什么是遗传算法？简述其基本思想和基本结构。

4．常用的适应度函数有哪几种？