第 2 章　与或图搜索问题

前一章介绍了状态空间搜索问题及有关的搜索算法。这些问题的特点是一个节点的后继节点之间是“或”的关系，只要一个节点得以求解，则该节点也就被求解了。从初始节点到目标节点之间，求解的是一条路径，也就是一个节点的序列。但在某些问题中，一个节点是否被求解，取决于该节点的部分或全部后继节点被求解，而不只是一个后继节点被求解。也就是说，对于该节点来说，其部分或全部后继节点是“与”的关系。这就是本章要讨论的与或图搜索问题。

2.1 与或图的搜索

在现实世界中，经常会遇到这样的问题：一个问题可以有几种求解方法，只要其中一种方法可以求解问题，则该问题被求解。也就是说，对求解该问题来说，方法之间是“或”的关系。但在用每一种方法求解时，又可能需要求解几个子问题，这些子问题必须全部求解，才可能用该方法求解原始问题。也就是说，这些子问题之间是“与”的关系。此类问题可以用与或图来表示。

通常我们要讨论与或图的一般情况（与或树是其特例）。在一个与或图中，一个节点可能会产生于不同的节点，比如节点 c 同时是节点 a 和节点 b 的后继节点。在这种情况下，可能会出现对于节点 a 来说 c 是“与”节点，而对于节点 b 来说，c 是“或”节点的情况发生。也就是说，一个节点是“与”节点还是“或”节点，是与其来源有关的。为了避免混淆不清，通常不把与或图中节点叫做与节点或者或节点，而是引入一个适合于与或图的更一般的标记，而在称谓上沿用习惯，仍把这种结构称作与或图。当然在讨论与或树时仍继续用“与”、“或”节点的称呼。

图 2.1 给出一个简单的与或图例子，下面就来说明它的标记方法。

这个图也称做超图，节点间是用超弧线（或连接符）来连接，如一个父节点和一组后继节点的关系可用若干个连接符来标记，并规定 k-连接符是从一个父节点指向一组 k 个后继节点的节点集。这时若节点间全是 1-连接符（相当于一条弧线）连接，显然这就是一般图的标记法，得到的就是与或图的特例——普通图。

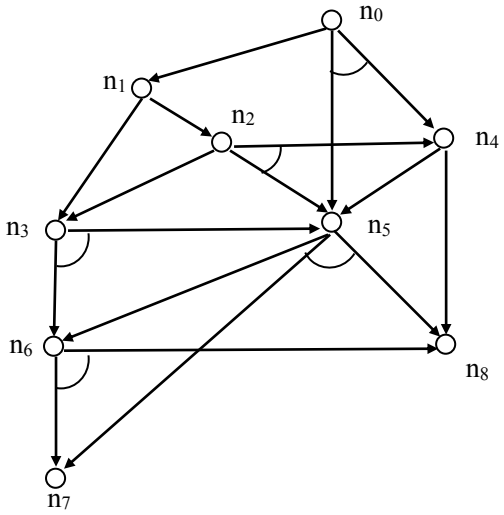


图 2.1 与或图

从图 2.1 可以看出，节点 n_0 有两个连接符：1-连接符指向节点 n_1；2-连接符指向节点集{n_4，n_5}。该 2-连接符还用一小段圆弧括起来（对 k＞1 的 k-连接符都应有小圆弧标记），以表示子节点间“与”的关系。可以看出这种标记法在节点间具有明确的关系。显然若用原先的术语，则对父节点 n_0 来说，n_4、n_5 是两个“与”节点，而 n_1 可称为“或”节点；而 n_8 既是 n_5 的一个“与”节点，又是 n_4 的一个“或”节点，混淆难于避免。另外也把图中没有任何父节点的节点叫根节点，没有任何后继节点的节点叫端节点或叶节点。

一个可分解产生式系统规定了一个隐含的与或图，初始数据库对应于图中的初始节点，它有一个外向的连接符连到它的一组后继节点，每个后继节点分别对应初始数据库中的一个分量；可应用于分量数据库的每条产生式规则，也对应于一个连接符，它指向的节点则相应于应用规则后生成的数据库；满足产生式系统结束条件的数据库是对应的一组终节点；产生式系统的任务是搜索从初始节点到一组终节点集 N 的一个解图。

下面就解图及其耗散值的概念和定义、能解不能解节点的定义作些说明。

与或图中某一个节点 n 到节点集 N 的一个解图类似于普通图中的一条解路径。解图的求法是：从节点 n 开始，正确选择一个外向连接符，再从该连接符所指的每一个后继节点出发，继续选一个外向连接符，如此进行下去直到由此产生的每一个后继节点成为集合 N 中的一个元素为止。图 2.2 给出 n_0→{n_7，n_8}的三个解图。

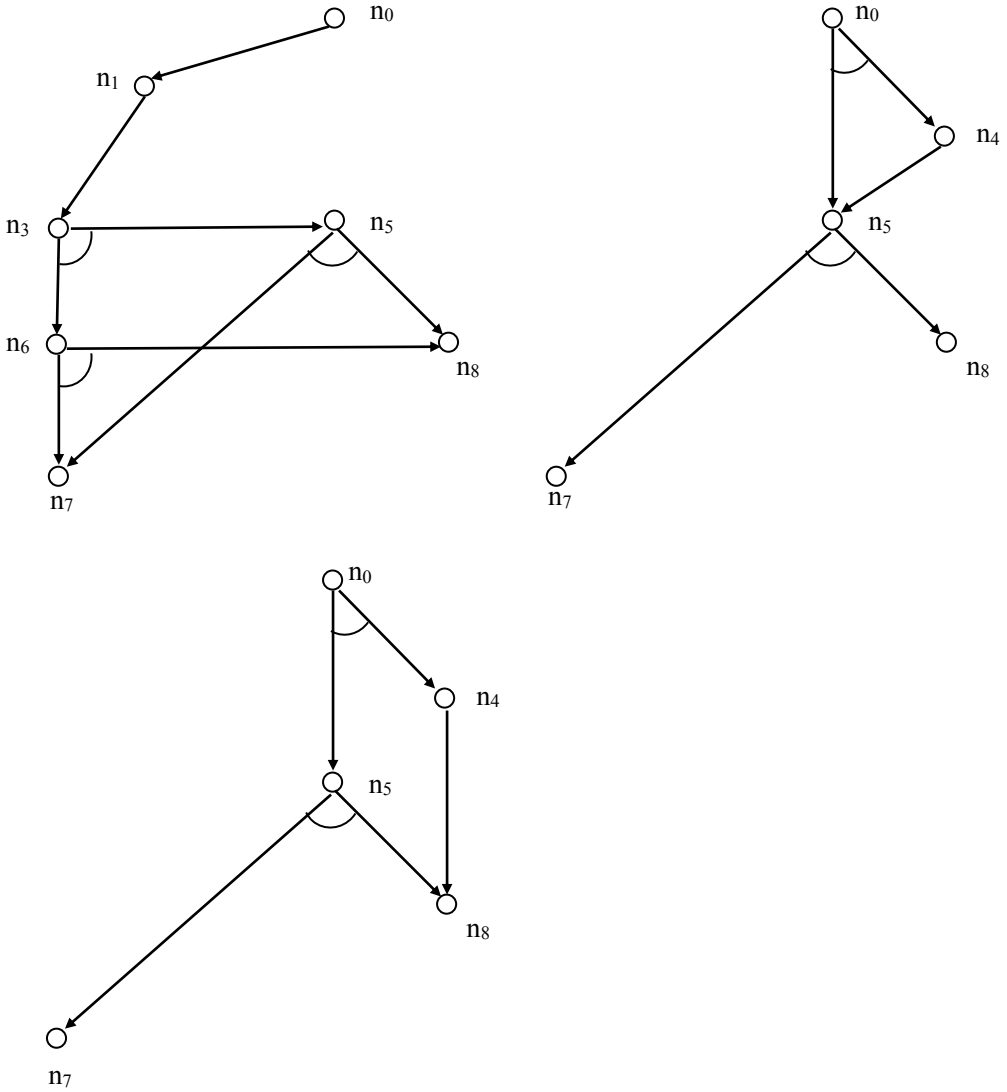


图 2.2 三个解图

在与或图是无环（即不存在这样的节点——其后继节后同时又是它的祖先）的假定下，解图可递归定义如下：

定义：一个与或图 G 中，从节点 n 到节点集 N 的解图记为G'，G'是 G 的子图。

②　n 是 N 的一个元素，则G'由单一节点组成；

②若 n 有一个指向节点{n_1，…，n_k}的外向连接符 K，使得从每一个 n_i 到 N 有一个解图（i=1，…，k），则G'由节点 n、连接符 K、及{n_1，…，n_k}中的每一个节点到 N 的解图所组成；

③否则 n 到 N 不存在解图。

同样我们可以递归定义局部图：

定义：

①单一节点是一个局部图；

②对于一个局部图的任意叶节点 n，选择一个 n 的外向连接符 K，则该局部图、外向连接符 K，以及 K 所连接的 n 的后继节点一起组成的图，仍然组成一个局部图。

在搜索解图的过程中，还须要进行耗散值的计算。若解图的耗散值记为 k（n，N），则可递归计算如下：

①若 n 是 N 的一个元素，则 k(n，N)＝0；

②若 n 是一个外向连接符指向后继节点$\{n_1，\cdots，n_i\}$，并设该连接符的耗散值为 C_n，则

$k(n，N)＝C_n+ k(n_1，N) + \cdots + k(n_i，N)$

根据这个定义，在假定 k 连接符的耗散值为 k 的情况下，图 2.2 三个解图的耗散值计算结果分别为 8、7 和 5。具有最小耗散值的解图称为最佳解图，其值也用 h*(n) 标记，上例中 $h^*(n_0)＝5$。

同样，也可以计算一个局部图的耗散值。如果我们同样将局部图的耗散值记为 k(n，N)，则：

② n 是局部图的一个叶节点，则 k(n，N)＝h(n)；

②若 n 有一个外向连接符指向后继节点$\{n_1，\cdots，n_i\}$，并设该连接符的耗散值为 C_n，则

$k(n，N)＝C_n+ k(n_1，N) + \cdots + k(n_i，N)$

其中，h(n)表示节点 n 到目标节点集的最佳解图耗散值的估计。

此外搜索过程还要标记能解节点(SOLVED)，为此给出如下定义：

能解节点(SOLVED)

①终节点是能解节点；

②若非终节点有“或”子节点时，当且仅当其子节点至少有一能解，该非终节点才能解；

③若非终节点有“与”子节点时，当且仅当其子节点均能解，该非终节点才能解。

不能解节点(UNSOLVED)；

①没有后裔的非终节点是不能解节点；

②若非终节点有“或”子节点时，当且仅当所有子节点均不能解时，该非终节点才不能解；

③若非终节点有“与”子节点时，当至少有一子节点不能解时，该非终节点才不能解。

2.2 与或图的启发式搜索算法 AO*

启发式的与或图搜索过程和普通图类似，也是通过评价函数 f(n)来引导搜索过程。

对于与或图来说，由于局部图中有多个叶节点，不能像普通图搜索那样，通过对某一个节点的评价来实现对整个局部图的评价。在普通图搜索中，f(n)=g(n)+h(n)，表面上是对 n 的评价，实际上是对通过 n 的这条路径的评价。因此对于与或图搜索来说，就是通过对局部图的评价来选择待扩展的节点。

下面首先讨论 AO*算法本身，然后再通过示例说明搜索过程以及与 A*算法的某些差别。

1．AO*算法

过程 AO*：

①建立一个搜索图 G，G:＝s，计算 q(s)＝h（s），IF GOAL（s）THEN M（s，SOLVED）；开始时图 G 只包括 s，耗散值估计为 h(s)，若 s 是终节点，则标记上能解。

②Until s 已标记上 SOLVED，do：

③begin

④ G':＝FIND(G)；根据连接符标记（指针）找出一个待扩展的局部解图 G'。

⑤ n:＝G'中的任一非终节点；选一个非终节点作为当前节点。

⑥ EXPAND(n)，生成子节点集{n_j}，计算 q(n_j)＝h(n_j)，其中 $n_j \notin G$，G:＝ADD({n_j}，G)，IF GOAL(n_j) THEN M(n_j，SOLVED)；对 G 中未出现的子节点计算耗散值，若有终节点则加能解标记，然后把 n 的子节点添加到 G 中。

⑦ S:＝{n}；建立含 n 的单一节点集合 S。

⑧ Until S 为空，do：

⑨ begin

⑩ REMOVE(m，S)，$m_c \notin S$；这个 m 的子节点 m_c 应不在 S 中。

⑪ ·修改 m 的耗散值 q(m)：

对 m 指向节点集{n_{1i}，n_{2i}，…，n_{ki}}的每一个连接符 i 分别计算 q_i：

$q_i(m)＝C_i+q(n_{1i})+\cdots+q(n_{ki})$，

q(m):＝min $q_i(m)$；对 m 的 i 个连接符，取计算结果最小的那个耗散值为 q(m)。

·加指针到 min $q_i(m)$的连接符上，或把指针修改到 min $q_i(m)$的连接符上，即原来指针与新确定的不一致时应删去。

·IF M(n_{ji}，SOLVED) THEN M(m，SOLVED)；若该连接符的所有子节点

都是能解的，则 m 也标上能解。

⑫ IF M(m，SOLVED)∨(q(m)≠q_0(m))

THEN ADD(ma，S)；m 能解或修正的耗散值与原先估算 q_0 不同，则把 m 的所有先辈节点 m_a 添加到 S 中。

⑬ end

⑭end

这个算法可划分成两个操作阶段：第一阶段是 4—6 步，完成自顶向下的图生成操作，先通过有标记的连接符，找到目前为止最好的一个局部解图，然后对其中一个非终节点进行扩展，并对其后继节点赋估计耗散值和加能解标记。第二阶段是 7—12 步，完成自底向上的耗散值修正计算、连接符（即指针）的标记以及节点的能解标记。

耗散值的修正从刚被扩展的节点 n 开始，其修正耗散值 q（n）取估计 h（n）的所有值中最小的一个，然后根据耗散值递归计算公式逐级向上修正其先辈节点的耗散值，只有下层节点耗散值修正后，才可能影响上一层节点的耗散值，因此必须自底向上一直修正到初始节点。这由算法中的内循环过程完成，下面就用图 2.1 的例子来说明算法的应用过程。

2．算法应用举例

设某个问题的状态空间如图 2.1 所示，并定义了某个启发函数 h（n），我们来看一看解图的搜索过程。

为了使用方便，将 h(n)函数对图 2.1 中各节点的假想估值先列写如下（实际应用中是节点生成出来之后才根据 h(n)定义式计算）：

$h(n_0)$＝3，$h(n_1)$＝2，$h(n_2)$＝4，$h(n_3)$＝4，$h(n_4)$＝1，$h(n_5)$＝1，$h(n_6)$＝2，$h(n_7)$＝$h(n_8)$＝0（目标节点）。

此外我们假设 k-连接符的耗散值为 k。

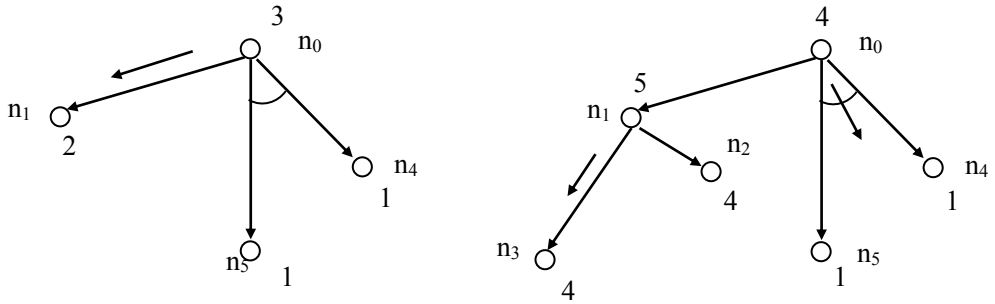
图 2.3 给出了使用 AO*算法对图 2.1 所示与或图的搜索过程。

开始时，只有一个初始节点 n_0，n_0 被扩展，生成出节点 n_1、n_4 和 n_5，一个 1－连接符指向 n_1，一个 2－连接符指向 n_4 和 n_5。这两个连接符之间是“或”的关系。由已知条件知：$h(n_1)$=2，$h(n_4)$=1，$h(n_5)$=1，并且已经假设 k－连接符的耗散值为 k。所以由 n_0 的 1－连接符，可以计算出 n_0 的耗散值（即算法中的 q 值，为了与算法一直，以下用 q 值表示）为（n_0 的 1－连接符的耗散值）＋（n_1 的 q 值）＝1＋2＝3。对于目前得到的局部图来说，n_1 是一个叶节点，所以其 q 值用 h 值（＝2）代替。由 n_0 的 2－连接符，可以计算出 n_0 的 q 值为（n_0 的 2－连接符的耗散值）＋（n_4 的 q 值）＋（n_5 的 q 值）＝2＋1＋1＝4。在两个不同渠道计算得到的耗散值中取最小值作为 n_0

的 q 值，并设置一个指针指向提供该耗散值的连接符。在这里 3<4，所以 n_0 的 q 值为 3，指针所指向的连接符为 n_0 的 1－连接符。至此算法的第一个循环结束。搜索图如图 2.3（a）所示。

在第二个循环中，首先从 n_0 开始，沿指针所指向的连接符，寻找一个未扩展的非终节点。这时找到的是 n_1 节点。扩展 n_1 节点，生成出节点 n_2 和 n_3，两个节点分别通过 1－连接符与 n_1 连接。由于 $h(n_2)=h(n_3)=4$，所以通过这两个连接符计算的耗散值也一样，均为 5。取其最小者还是 5，从而更新 n_1 的 q 值为 5。由于耗散值相等，所以指向连接符的指针可以指向两个连接符中的任何一个，假定指向了 n_3 这一边。由于 n_1 的 q 值由 2 更新为 5，所以需要从新计算 n_0 的 q 值。对于 n_0 来说，此时从 1－连接符这边算得的耗散值为 6，大于从 2－连接符这边得到的耗散值 4，所以 n_0 的 q 值更新为 4，并将指向连接符的指针由指向 1－连接符改为指向 2－连接符。注意，这时由 n_1 出发的指向连接符的指针并没有被改变或删除。至此第二循环结束，搜索图如图 2.3（b）所示。

在第三个循环中，同样从 n_0 开始，沿指针所指向的连接符，寻找未扩展的非终节点。这时 n_4 和 n_5 都是满足这一条件的节点，可以从它们中任选择一个扩展。假定选择的是 n_5。n_5 生成出 n_6、n_7 和 n_8 三个节点。一个 1－连接符指向 n_6，一个 2－连接符指向 n_7 和 n_8。计算的结果，得到 n_5 的 q 值为 2，指针指向 2－连接符。由于 n_5 的 q 值改变了，因此需要重新计算 n_0 的 q 值，计算的结果为 5。该结果还是比通过 n_0 的 1－连接符计算得到的耗散值小，因此只需更新 n_0 的 q 值即可，不需要改变指向连接符的指针。



(a)　　(b)

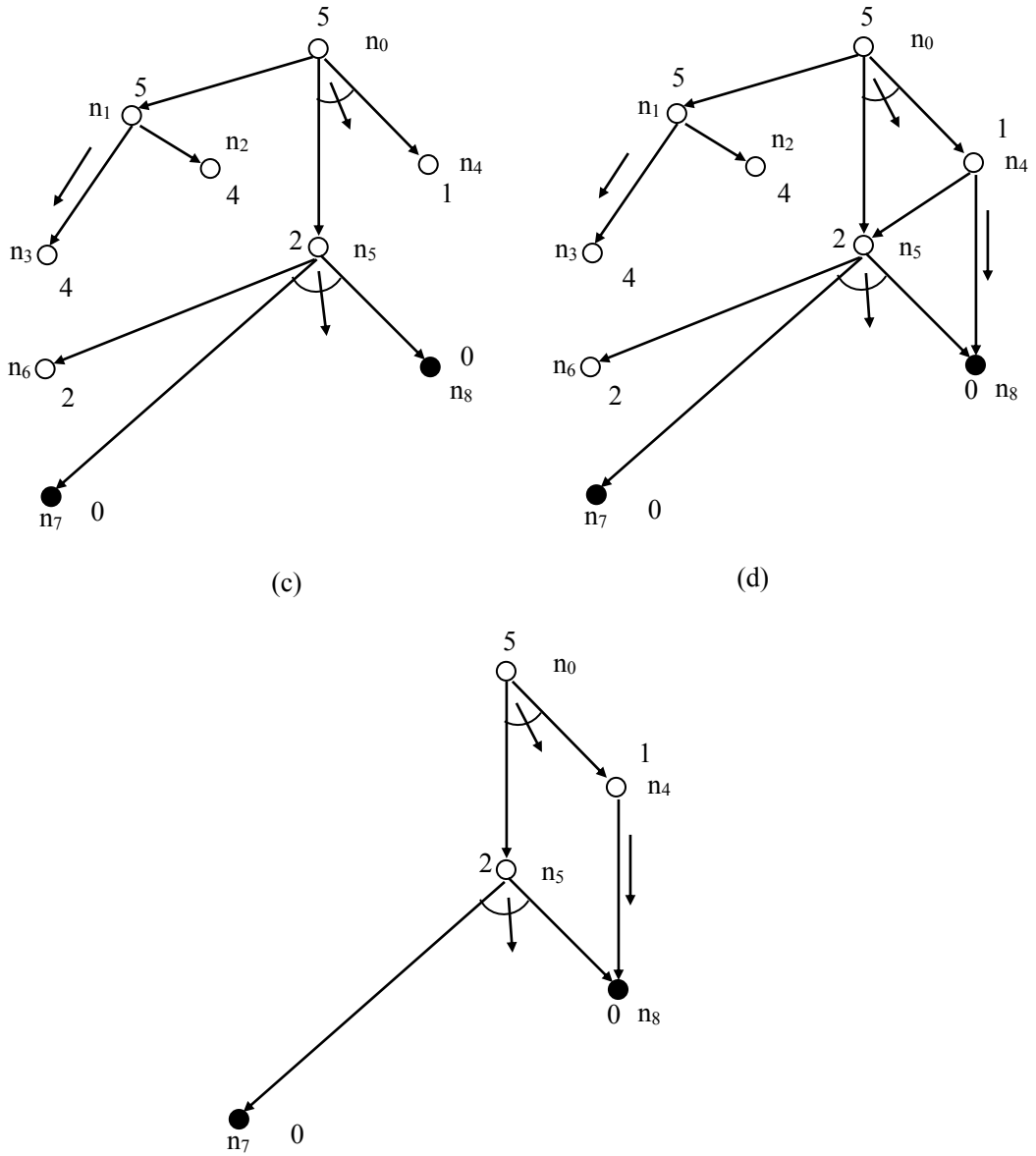


图 2.3 AO*算法 4 个循环的搜索图及搜索得到的解图

（a）一次循环之后（b）两次循环之后

（c）三次循环之后（d）四次循环之后

（e）搜索得到的解图

在本次循环中，由于 n_7 和 n_8 都是目标节点，是能解节点，而 n_5 通过一个 2－连接符连接 n_7 和 n_8，所以 n_5 也被标记为能解节点。虽然 n_5 是能解节点，但由于 n_0 是通过一个 2－连接符连接 n_5 和 n_4，而此时 n_4 还不是能解的，所以 n_0 还不是能解的。搜

索将继续进行。至此第三循环结束，搜索图如图 2.3（c）所示。

第四次循环，同样的道理，从 n_0 开始沿指针所指向的连接符，寻找未扩展的非终节点，这次找到的是 n_4。扩展 n_4，生成节点 n_5 和 n_8，并分别以 1－连接符连接。对 n_4 来说，从 n_5 这边计算得耗散值为 3（n_5 已经被扩展过，其 q 值已经被更新为 2，因此在计算 n_4 的耗散值时，应使用更新后的 q 值，而不是 n_5 的 h 值），从 n_8 这边计算得耗散值为 1。取小者，得 n_4 的 q 值为 1，并且将指向连接符的指针指向 n_8 这边的连接符。因为扩展 n_4 并没有改变它的 q 值，因此 n_0 的 q 值也不需重新计算了。由于 n_8 是目标节点，是能解节点，而 n_4 通过一个 1－连接符连接 n_8，因此 n_4 也被标记为能解节点。在第三循环中，n_5 已经被标记为能解节点。这样 n_4、n_5 都是能解节点了，从而 n_0 也被标记为能解节点。而 n_0 是初始节点，至此搜索全部结束。从 n_0 开始，沿指向连接符的指针找到的解图即为搜索的结果。图 2.3（e）为得到的解图。

3．算法应用的若干问题

（1）在第 6 步扩展节点 n 时，若不存在后继节点（即陷入死胡同），则可在第 11 步中对 m（即 n）赋一个高的 q 值，这个高的 q 值会依次传递到 s，使得含有节点 n 的子图具有高的 q（s），从而排除了被当作候选局部解图的可能性。

（2）第 5 步中怎样选出 G′ 中的一个非终节点来扩展呢？一般可以选一个最可能导致该局部解图耗散值发生较大变化的那个节点先扩展，因为选这个节点先扩展，会促使及时修改局部解图的标记。

（3）同 A 算法类似，若 s→N 集存在解图，当 h（n）≤h*（n）且 h（n）满足单调限制条件时，则 AO*一定能找到最佳解图，即 AO*具有可采纳性。当 h（n）≡0 时，AO*也蜕化为宽度优先算法。

这里单调限制条件是指：对隐含图中，从节点 $n\rightarrow\{n_1, \cdots, n_k\}$的每一个连接符都施加限制，即假定 $h(n)\leqslant C+h(n_1)+\cdots+h(n_k)$ 其中 C 是连接符的耗散值。

如果 $h(n)$ 满足单调限制，且 $h(t_i)=0$（$t_i\in N$），那么单调限制意味着 h 是 h* 的下界范围，即对所有节点 n 有 $h(n)\leqslant h^*(n)$。

综上所述，看出 AO*与 A 有类似之处，但也有若干区别。首先是 AO*算法不能像 A 算法那样，单纯靠评价某一个节点来评价局部图。其次是由于 k-连接符连接的有关子节点，对父节点能解与否以及耗散值都有影响，因而显然不能象 A 算法那样优先扩展其中具有最小耗散值的节点。再一点是 AO*算法仅适用于无环图的假设，否则耗散值递归计算不能收敛，因而在算法中还必须检查新生成的节点已在图中时，是否是正被扩展节点的先辈节点。最后还必须注意 A 算法设有 OPEN 和 CLOSED 表，而 AO*算法只用一个结构 G，它代表到目前为止已明显生成的部分搜索图，图中每一

个节点的 h（n）值是估计最佳解图，而不是估计解路径。

有关 AO*算法的具体应用及一些提高搜索效率的修正算法，可进一步参阅人工智能有关文献。

2.3 博弈树的搜索

这里所讲的博弈，指的是类似于象棋这样的游戏问题。这类问题有以下一些特点：（1）双人对弈，对垒的双方轮流走步。（2）信息完备，对垒双方所得到的信息是一样的，不存在一方能看到，而另一方看不到的情况。（3）零和。即对一方有利的棋，对另一方肯定是不利的，不存在对双方均有利、或均无利的棋。对弈的结果是一方赢，而另一方输，或者双方和棋。

博弈问题为什么可以用与或图表示呢？可以这样来看待这个问题：当轮到我方走棋时，只需从若干个可以走的棋中，选择一个棋走就可以了。从这个意义上说，若干个可以走的棋是“或”的关系。而对于轮到对方走棋时，对于我方来说，必须能够应付对手的每一种走棋。这就相当于这些棋是“与”的关系。因此，博弈问题可以看成是一个与或图，但是与一般的与或图并不一样，是一种特殊的与或图。

1．概述

博弈一向被认为是富有智能行为的游戏，因而很早就受到人工智能界的重视，早在 60 年代就已经出现若干博弈程序，并达到一定水平。博弈问题的研究还不断提出一些新的研究课题，从而也推动了人工智能研究的发展。

对于单人博弈的一些问题，可用一般的搜索技术进行求解，本节着重讨论双人完备信息这一类博弈问题的搜索策略。由于双人博弈可用与或树（图）来描述，因而搜索就是寻找最佳解树（图）的问题。

所谓双人完备信息，就是两位选手对垒，轮流走步，这时每一方不仅都知道对方过去已经走过的棋步，而且还能估计出对方未来可能的走步。对弈的结果是一方赢（另一方则输），或者双方和局。这类博弈的实例有：一字棋、余一棋、西洋跳棋、国际象棋、中国象棋、围棋等。对于带机遇性的任何博弈，因不具有完备信息，不属这里讨论范围，但有些论述可推广到某些机遇博弈中应用。

博弈问题可以用产生式系统的形式来描述，例如中国象棋，综合数据库可规定为棋盘上棋子各种位置布局的一种描述，产生式规则是各类棋子合法走步的描述，目标则可规定为将（帅）被吃掉，规则作用于数据库的结果便生成出博弈图或博弈树。下面举一个简单的例子说明博弈问题可用与或图表示，并讨论搜索策略应考虑的实际问

题。

2．Grundy 博弈

Grundy 博弈是一个分钱币的游戏。有一堆数目为 N 的钱币，由两位选手轮流进行分堆，要求每个选手每次只把其中某一堆分成数目不等的两小堆。例如选手甲把 N 分成两堆后，轮到选手乙就可以挑其中一堆来分，如此进行下去直到有一位选手先无法把钱币再分成不相等的两堆时就得认输。

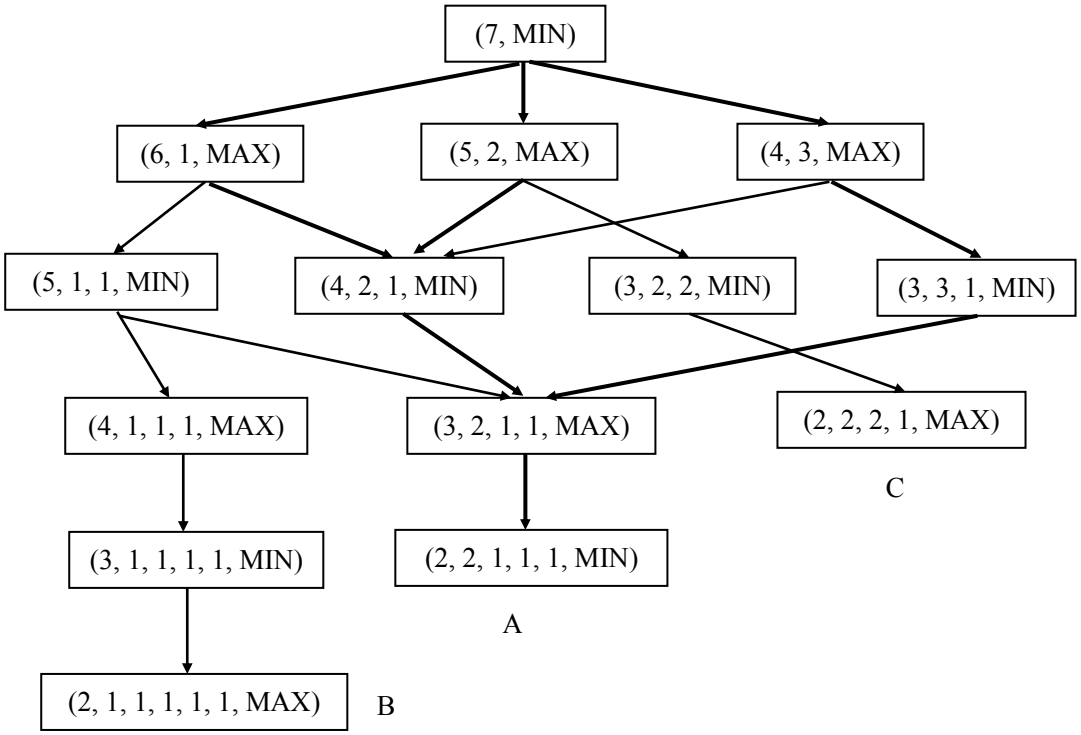


图 2.4 Grundy 博弈状态空间图

设初始状态为（7，MIN），则该问题的状态空间图如图 2.4 所示，图中所有终节点均表示该选手必输的情况，取胜方的目标是设法使棋局发展为结束在对方走步时的终节点上，因此节点 A 是 MAX 的搜索目标，而节点 B，C 则为 MIN 的搜索目标。

搜索策略要考虑的问题是：

对 MIN 走步后的每一个 MAX 节点，必须证明 MAX 对 MIN 可能走的每一个棋局对弈后能获胜，即 MAX 必须考虑应付 MIN 的所有招法，这是一个与的含意，因此含有 MAX 符号的节点可看成与节点。

对 MAX 走步后的每一个 MIN 节点，只须证明 MAX 有一步能走赢就可以，即 MAX 只要考虑能走出一步棋使 MIN 无法招架就成，因此含有 MIN 符号的节点可看成或节点。

这样一来对弈过程的搜索图就呈现出与或图表示的形式，从搜索图中可以看出，MAX 存在完全取胜的策略，图中粗线所示部分就代表了这种策略，这时不论 MIN 怎么走，MAX 均可取胜。因而寻找 MAX 的取胜策略便和求与或图的解图一致起来，即 MAX 要取胜，必须对所有与节点取胜，但只需对一个或节点取胜，这就是一个解图。因此实现一种取胜的策略就是搜索一个解图的问题，解图就代表一种完整的博弈策略。

对于 Grundy 这种较简单的博弈，或者复杂博弈的残局，可以用类似于与或图的搜索技术求出解图，解图代表了从开局到终局任何阶段上的弈法。显然这对许多博弈问题是不可能实现的。例如中国象棋，每个势态有 40 种不同的走法，如果一盘棋双方平均走 50 步，则搜索的位置有（40^2）$^{50}\approx10^{160}$，即深度达 100 层，总节点数约为 10^{161} 个。因此要考虑完整的搜索策略，即便是用当前最快的计算机来处理，也得花天文数字计的时间。对于西洋跳棋、国际象棋大致也如此，而围棋就更复杂了。因此即使用了强有力的启发式搜索技术，也不可能使分枝压到很少，因此这种完全取胜策略（或和局）必须丢弃，而应当把目标确定为寻找一步好棋，等对手回敬后再考虑寻找另一步好棋这种实际可行的实用策略。这种情况下每一步结束条件可根据时间限制、存储空间限制或深度限制等因素加以确定。搜索策略可采用宽度、深度或启发式方法，一个阶段搜索结束后，要从搜索树中提取一个优先考虑的“最好的”走步，这就是实用策略的基本点。下面就来讨论这种机理的搜索策略——极小极大搜索过程。

3．极小极大搜索过程

极小极大搜索方法是博弈树搜索的基本方法，现在博弈树搜索中最常用的 α - β 剪枝搜索方法，就是从这一方法发展而来的。

首先假定，有一个评价函数可以对所有的棋局进行评估。当评价函数值大于 0 时，表示棋局对我方有利，对对方不利。当评价函数小于 0 时，表示棋局对我方不利，对对方有利。而评价函数值越大，表示对我方越有利。当评价函数值等于正无穷大时，表示我方必胜。评价函数值越小，表示对我方越不利。当评价函数值等于负无穷大时，表示对方必胜。假设双方都是对弈高手，在只看一步棋的情况下，我方一定走评价函数值最大的一步棋，而对方一定走评价函数值最小的一步棋。会下棋的读者都知道，在只看一步的情况下最好的棋，从全局来说不一定就好，还可能很不好。因此为了走出好棋，必须多看几步，从多种可能状态中选择一步好棋。

想一想人是如何下棋的呢？人实际上采用的是一种试探性的方法。首先假定走了一步棋，看对方会有那些应法，然后再根据对方的每一种应法，看我方是否有好的回应.....这一过程一直进行下去，直到若干步以后，找到了一个满意的走法为止。初学者可能只能看一、两个轮次，而高手则可以看几个，甚至十几个轮次。

极小极大搜索方法，模拟的就是人的这样一种思维过程。

极小极大搜索策略是考虑双方对弈若干步之后，从可能的走步中选一步相对好棋的着法来走，即在有限的搜索深度范围内进行求解。为此要定义一个静态估计函数 f，以便对棋局的势态（节点）作出优劣估值，这个函数可根据势态优劣特征来定义（主要用于对端节点的“价值”进行度量）。一般规定有利于 MAX 的势态，f（p）取正值，有利于 MIN 的势态，f（p）取负值，势均力敌的势态，f（p）取 0 值。若 f（p）＝＋∞，则表示 MAX 赢，若 f（p）＝－∞，则表示 MIN 赢。下面的讨论规定：顶节点深度 d＝0，MAX 代表程序方，MIN 代表对手方，MAX 先走。

图 2.5 是一个表示考虑两步棋的例子，图中端节点给出的数字是用静态函数 f（p）计算得到，其他节点不用 f（p）估计，因为不够精确，而应用倒推的办法取值。例如 A、B、C 是 MIN 走步的节点，MAX 应考虑最坏的情况，故其估值应分别取其子节点 f（p）估值中最小者，而 s 是 MAX 走步的节点，可考虑最好的情况，故估值应取 A、B、C 值中的最大者。这样求得 f（s）＝－2，依此确定了相对较优的走步应是走向 B，因为从 B 出发，对手不可能产生比 f（s）＝－2 更差的效果。实际上可根据资源条件，考虑更多层次的搜索过程，从而可得到更准确的倒推值，供 MAX 选取更正确的走步。当用端节点的静态估计函数 f（p）求倒推值时，两位选手应采取不同的策略，自底向上逐层交替使用极小和极大的选值方法，故称极小极大过程。

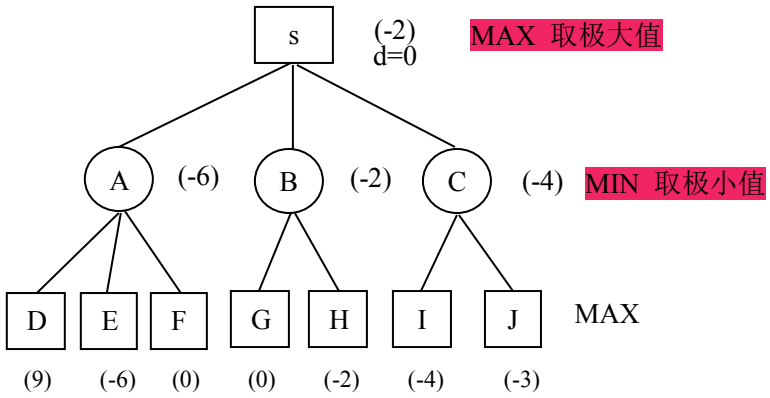


图 2.5 f(p)求值过程

过程 MINIMAX：

①T:＝(s，MAX)，OPEN:＝(s)，CLOSED:＝(　)；开始时树由初始节点构成，OPEN 表只含有 s。

②LOOP1:IF OPEN＝(　)THEN GO LOOP2；

③n:＝FIRST(OPEN)，REMOVE(n，OPEN)，ADD(n，CLOSED)；

④IF n 可直接判定为赢、输或平局

THEN f(n):＝∞∨－∞∨0，GO LOOP1

ELSE EXPAND(n)→{n_i}，ADD({n_i}，T)

IF d(n_i)＜k THEN ADD({n_i}，OPEN)，GO LOOP1

ELSE 计算 f(n_i)，GO LOOP1；n_I 达到深度 k，计算各端节点 f 值。

⑤LOOP2：IF CLOSED＝NIL THEN GO LOOP3

ELSE n_p:＝FIRST(CLOSED)；

⑥IF(n_p∈MAX)∧(f(n_{ci}∈MIN)有值)

THEN f(n_p):＝max{f(n_{ci})}，REMOVE(n_p，CLOSED)；若 MAX 所有子节点均有值，则该 MAX 取其极大值。

IF (n_p∈MIN)∧(f(n_{cj}∈MAX)有值)

THEN f(n_p):＝min{f(n_{cj})}，REMOVE(n_p，CLOSED)；若 MIN 所有子节点均有值，则该 MIN 取其极小值。

⑦GO LOOP2；

⑧LOOP3：IF f（s）≠NIL THEN EXIT(END∨M(Move，T))；s 有值，则结束或标记走步。

该算法分两个阶段进行，第一阶段②—④是用宽度优先法生成规定深度的全部博弈树，然后对其所有端节点计算其静态估计函数值。第二阶段⑥—⑧是自底向上逐级求非端节点的倒推估计值，直到求出初始节点的倒推值 f（s）为止，此时

$$f(s)=\max_{i_1}\min_{i_2}\Lambda\{f(n_{i_1 i_2\Lambda i_k})\}$$

其中 $n_{i_1 i_2 .. i_k}$ 表示深度为 k 的端节点。再由 f（s）即可选得相对较好的走步来，过程遂造结束。等对手响应走步后，再以当前的状态作为起始状态 s，重复调用该过程。

下面通过最简单的 3×3 棋盘的一字棋为例来说明算法的应用过程。

在九宫格棋盘上，两位选手轮流在棋盘上摆各自的棋子（每次一枚），谁先取得三子一线的结果就取胜。

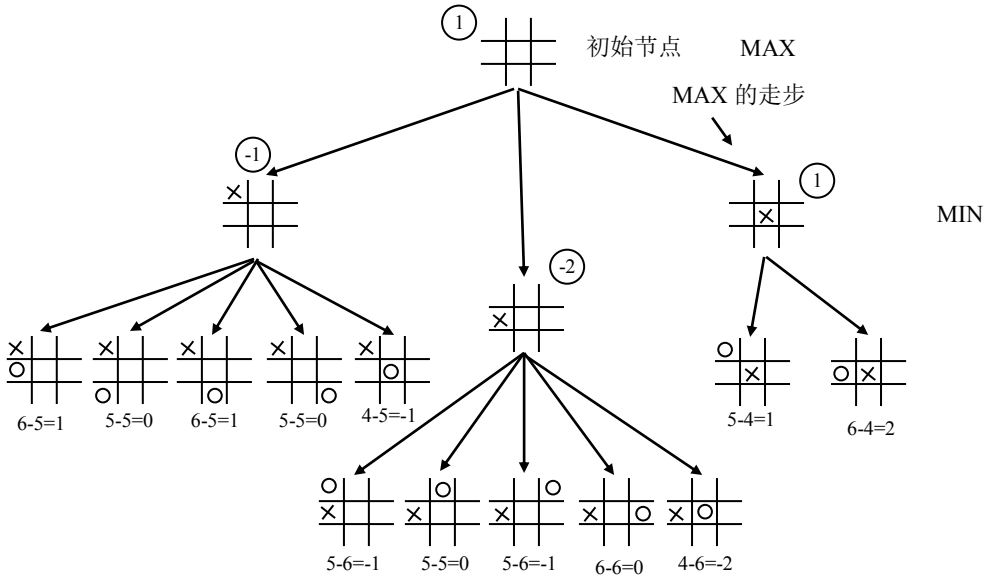


图 2.6 一字棋第一阶段搜索树

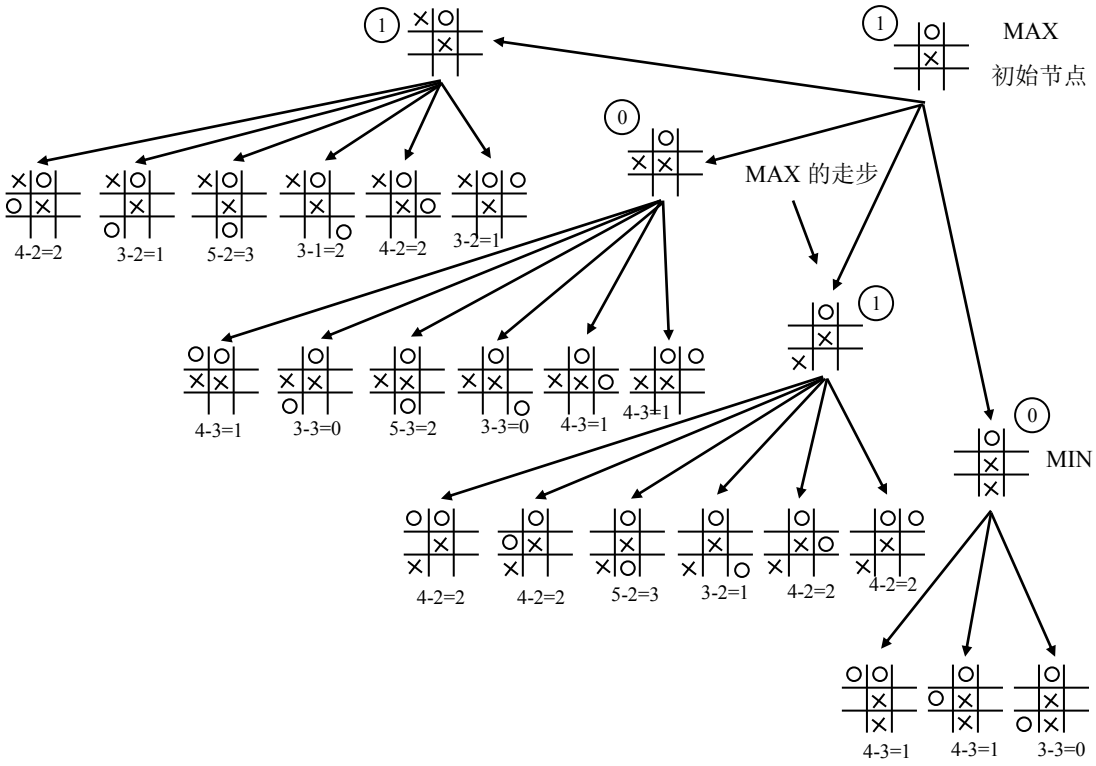


图 2.7 一字棋第二阶段搜索树

设程序方 MAX 的棋子用（×）表示，对手 MIN 的棋子用（○）表示，MAX 先走。静态估计函数 f(p)规定如下：

若 p 对任何一方来说都不是获胜的格局，则

f(p)＝（所有空格都放上 MAX 的棋子之后，MAX 的三子成线（行、列、对角）的总数）－（所有空格都放上 MIN 的棋子之后，MIN 的三子成线（行、列、对角）的总数）

若 p 是 MAX 获胜的格局，则 f(p)＝∞；

若 p 是 MIN 获胜的格局，则 f(p)＝－∞。

例如，当 p 的格局如下时，则可得 f(p)＝6－4＝2；

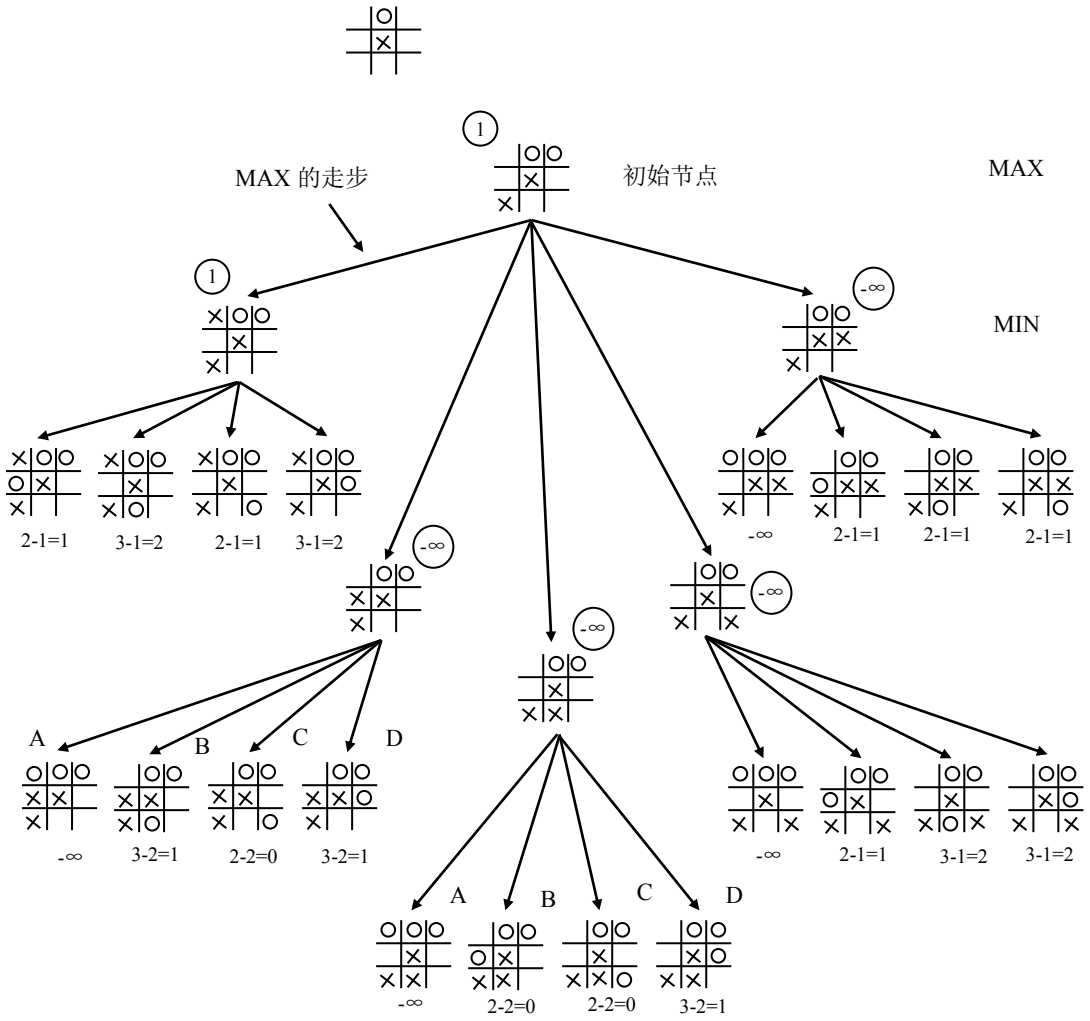


图 2.8 一字棋第三阶段搜索树

设考虑走两步的搜索过程，利用棋盘对称性的条件，则第一次调用算法产生的搜索树如图 2.6 所示，图中端节点下面是计算的 f(p)值，非端节点的倒推值标记在圆圈内。为了使初始节点具有最大的倒推值，可以看出第一步的最好着法是把棋子下在中央位置。

设 MAX 走完第一步后，MAX 的对手是在×之上的格子下子，这时 MAX 就要在新格局下调用算法，第二次产生的搜索树如图 2.7 所示。类似的第三次的搜索树如图 2.8 所示。至此 MAX 走完最好的走步后，不论 MIN 怎么走，都无法挽回败局，因此只好认输了，否则还要进行第四轮回的搜索。

4. α－β 搜索过程

MINIMAX 过程是把搜索树的生成和格局估值这两个过程分开来进行，即先生成全部搜索树，然后再进行端节点静态估值和倒推值计算，这显然会导致低效率。如图 2.8 中，其中一个 MIN 节点要全部生成 A、B、C、D 四个节点，然后还要逐个计算其静态估值，最后在求倒推值阶段，才赋给这个 MIN 节点的倒推值－∞。其实，如果生成节点 A 后，马上进行静态估值，得知 f（A）＝－∞之后，就可以断定再生成其余节点及进行静态计算是多余的，可以马上对 MIN 节点赋倒推值－∞，而丝毫不会影响 MAX 的最好优先走步的选择。这是一种极端的情况，实际上把生成和倒推估值结合起来进行，再根据一定的条件判定，有可能尽早修剪掉一些无用的分枝，同样可获得类似的效果，这就是 α - β 过程的基本思想。下面再用一字棋的例子来说明 α - β 剪枝方法。

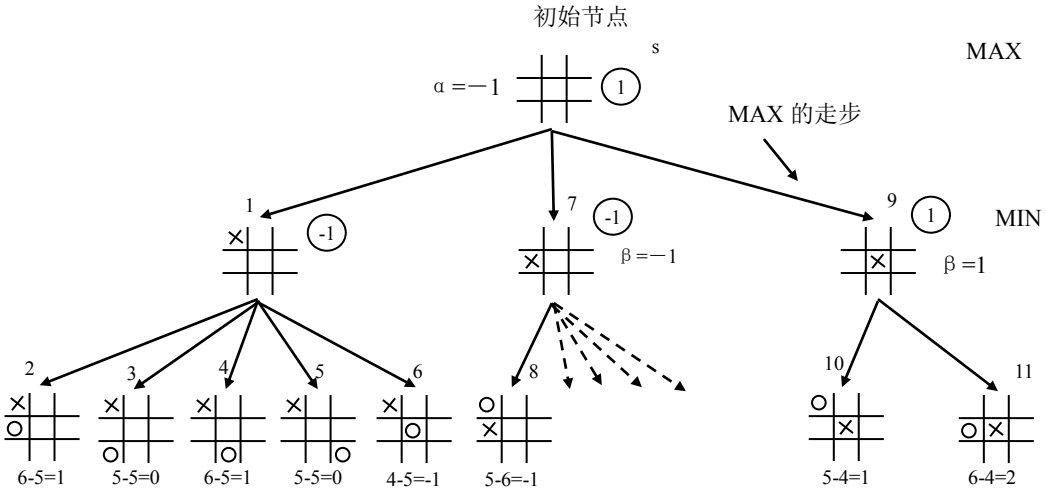


图 2.9 一字棋第一阶段 α - β 剪枝方法

为了使生成和估值过程紧密结合，采用有界深度优先策略进行搜索，这样当生成达到规定深度的节点时，就立即计算其静态估值函数，而一旦某个非端节点有条件确定其倒推值时就立即计算赋值。从图 2.9 中标记的节点生成顺序号（也表示节点编号）看出，生成并计算完第 6 个节点后，第 1 个节点倒推值完全确定，可立即赋给倒推值－1。这时对初始节点来说，虽然其他子节点尚未生成，但由于 s 属极大值层，可以推断其倒推值不会小于－1，我们称极大值层的这个下界值为α，即可以确定 s 的α＝－1。这说明 s 实际的倒推值决不会比－1 更小，还取决于其他后继节点的倒推值，因此继续生成搜索树。当第 8 个节点生成出来并计算得静态估值为－1 后，就可以断定第 7 个节点的倒推值不可能大于－1，我们称极小值层的这个上界值为β，即可确定节点 7 的β＝－1。有了极小值层的β值，很容易发现若α≥β时，节点 7 的其他子节点不必再生成，这不影响高一层极大值的选取，因 s 的极大值不可能比这个β值还小，再生成无疑是多余的，因此可以进行剪枝。这样一来，只要在搜索过程记住倒推值的上下界并进行比较，就可以实现修剪操作，称这种操作为α剪枝。类似的还有β剪枝，统称为α-β剪枝技术。在实际修剪过程中，α、β还可以随时修正，但极大值层的倒推值下界α永不下降，实际的倒推值取其后继节点最终确定的倒推值中最大的一个倒推值。而极小值层的倒推值上界β永不上升，其倒推值则取后继节点最终确定的倒推值中最小的一个倒推值。

归纳一下以上讨论，可将α-β过程的剪枝规则描述如下：

（1）α剪枝：若任一极小值层节点的β值小于或等于它任一先辈极大值居节点的α值，即α(先辈层)≥β(后继层)，则可中止该极小值层中这个 MIN 节点以下的搜索过程。这个 MIN 节点最终的倒推值就确定为这个β值

（2）β剪枝：若任一极大值层节点的α值大于或等于它任一先辈极小值层节点的β值，即α(后继层)≥β(先辈层)，则可以中止该极大值层中这个 MAX 节点以下的搜索过程。这个 MAX 节点的最终倒推值就确定为这个α值。

在进行α-β剪枝时，应注意以下几个问题：（1）比较都是在极小节点和极大节点间进行的，极大节点和极大节点的比较，或者极小节点和极小节点间的比较是无意义的。（2）在比较时注意是与“先辈层”节点比较，不只是与父辈节点比较。当然，这里的“先辈层”节点，指的是那些已经有了值的节点。（3）当只有一个节点的值“固定”以后，其值才能够向其父节点传递。（4）α-β剪枝方法搜索得到的最佳走步与极小极大方法得到的结果是一致的，α-β剪枝并没有因为提高效率，而降低得到最佳走步的可能性。（5）在实际搜索时，并不是先生成指定深度的搜索图，再在搜索图上进行剪枝。如果这样，就失去了α-β剪枝方法的意义。在实际程序实现时，首先规定一个搜索深度，然后按照类似于深度优先搜索的方式，生成节点。在节点的生成

过程中，如果在某一个节点处发生了剪枝，则该节点其余未生成的节点就不再生成了。

根据这些剪枝规则，很容易给出 α - β 算法描述，显然剪枝后选得的最好优先走步，其结果与不剪枝的 MINIMAX 方法所得完全相同，因而 α - β 过程具有较高的效率。

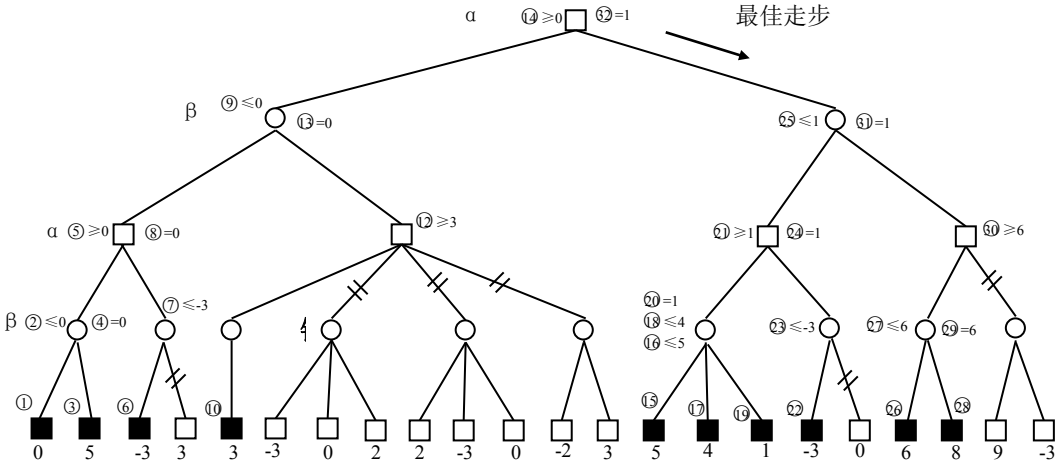


图 2.10 α - β 搜索过程的博弈树

图 2．10 给出一个 d＝4 的博弈树的 α - β 搜索过程，其中带圆圈的数字表示求静态估值及倒推值过程的次序编号。该图详细表示出 α - β 剪枝过程的若干细节。初始节点的最终倒推值为 1，该值等于某一个端节点的静态估值。最好优先走步是走向右分枝节点所代表的棋局，要注意棋局的发展并不一定要沿着通向静态值为 1 的端节点这条路径走下去，这要看对手的实际响应而定。

通过对图 2.10 的搜索，来说明 α - β 剪枝搜索过程。

在搜索过程中，假定节点的生成次序是从上到下，从左到右进行的。图中带圈的数字，表示节点的计算次序，在叙述时，为了表达上的方便，该序号也同时表示节点。当一个节点有两个以上的序号时，不同的序号，表示的是同一个节点在不同次序下计算的结果。

过程如下：

首先，从根节点开始，向下生成出到达指定节点深度的节点①，由①的值为 0，可知②≤0，继续扩展生成节点③，由于③的值 5 大于②的值 0，并且节点②再也没有其它的子节点了，所以④（与②是同一个节点）的值确定为 0。由④的值，可以确定⑤≥0。扩展节点⑤，按顺序生成节点⑦、⑥，由⑥的值-3，有⑦≤-3。⑦是极小节点，其值小于它的先辈节点⑤的值，满足 α 剪枝条件，故⑦的其他子节点被剪掉，不再生成。由于⑤不再有其他子节点，所以有⑧＝0，并因此有⑨≤0。扩展节点⑨，顺

序生成节点⑫、⑪、⑩，由⑩＝3 得到⑪＝3 和⑫≥3。⑫是极大节点，其值大于其先辈节点⑨的值，满足β剪枝条件，故⑫的其他三个子节点及其这些子节点的后继节点全部被剪掉。这样⑨的子节点也全部搜索完，得到⑬＝0，并上推到⑭≥0。扩展⑭的另一个子节点，一直到指定深度。由⑬＝5，有⑯≤5，然后顺序生成出⑰、⑲，由⑰、⑲的值分别得到⑱≤4、⑳＝1。上推得到㉑≥1。扩展㉑的另一个子节点，顺序生成㉓、㉒，得到㉓≤-3。㉓是一个极小节点，其值小于其先辈节点㉑的值，满足α剪枝条件，所以在㉓处发生剪枝（注意，此处即便是㉓的值不小于㉑的值，但由于㉓的值小于⑭的值，同样也发生α剪枝，因为是后继节点的β值与其先辈节点的α值进行比较，而不只是后辈的β值与其父节点的α值进行比较。同样，在β剪枝的情况下，也是后辈节点的α值与其先辈节点的β值进行比较，而不只是后辈的α值与其父节点的β值进行比较）。这样得到㉔＝1，㉕≤1。扩展㉕的右边子节点及其后继节点，有㉖＝6，㉗≤6，㉘＝8，㉙＝6，并由此上推到㉚≥6。㉚是一个极大节点，其值大于其先辈节点㉕的值，满足β剪枝条件，故在节点㉚处发生剪枝。这样使得㉛＝1，并使得根节点㉜＝1。至此全部搜索结束，根节点㉜的值就是对当前棋局的评判。由于该值来自于根节点的右边一个子节点㉛，所以搜索得到的最佳走步应该走向根节点的右边这一个子节点㉛。

应该注意的是，博弈树搜索的目标就是找到当前棋局的一步走法，所以α-β剪枝搜索的结果是得到了一步最佳走步，而不是象一般的图搜索或者与或图搜索那样，得到的是从初始节点到目标节点（集）的一条路径或者解图。

下面分析一下剪枝的效率问题。若以最理想的情况进行搜索，即对 MIN 节点先扩展最低估值的节点（若从左向右顺序进行，则设节点估计值从左向右递增排序），MAX 先扩展最高估值的节点（设估计值从左向右递减排序），则当搜索树深度为 D，分枝因数为 B 时，若不使用α-β剪枝技术，搜索树的端节点数 $N_D=B^D$；若使用α-β剪枝技术．可以证明理想条件下生成的端节点数最少，有

$N_D=2B^{D/2}-1$（D 为偶数）

$N_D=B^{(D+1)/2}+B^{(D-1)/2}-1$（D 为奇数）

比较后得出最佳α-β搜索技术所生成深度为D处的端节点数约等于不用α-β搜索技术所生成深度为D／2处的端节点数。这就是说，在一般条件下使用α-β搜索技术，在同样的资源限制下，可以向前考虑更多的走步数，这样选取当前的最好优先走步，将带来更大的取胜优势。

5．其他改进方法

使用α-β剪枝技术，当不满足剪枝条件（即α≥β）时。若β值比α值大不了

多少或极相近，这时也可以进行剪枝，以便有条件把搜索集中到会带来更大效果的其他路径上，这就是中止对效益不大的一些子村的搜索，以提高搜索效率。

其他改善极小极大过程性能的基本方法有：

（1）不严格限制搜索的深度，当到达深度限制时，如出现博弈格局有可能发生较大变化时（如出现兑子格局），则应多搜索几层，使格局进入较稳定状态后再中止，这样可使倒推值计算的结果比较合理，避免考虑不充分产生的影响，这是等候状态平稳后中止搜索的方法。

（2）当算法给出所选的走步后，不马上停止搜索，而是在原先估计可能的路径上再往前搜索几步，再次检验会不会出现意外，这是一种增添辅助搜索的方法。

（3）对某些博弈的开局阶段和残局阶段，往往总结有一些固定的对弈模式，因此可以利用这些知识编好走步表，以便在开局和结局时使用查表法。只是在进入中盘阶段后，再调用其他有效的搜索算法，来选择最优的走步。

α - β 剪枝过程是二人博弈问题的一般搜索方法，实践证明，该方法可以有效地减少在搜索过程中生成的节点数，提高搜索效率。IBM 公司研制的“深蓝”国际象棋程序，采用的就是这样的一种搜索算法，该程序曾经战胜了国际象棋世界冠军卡斯帕罗夫。

以上介绍的各种博弈搜索技术可用于求解所提到的一些双人博弈问题。但是这些方法还不能全面反映人们弈棋过程实际所使用的一切推理技术，也未涉及棋局的表示和启发函数问题。例如一些高明的棋手，对棋局的表示有独特的模式，他们往往记住的是一个可识别的模式集合，而不是单独棋子的具体位置。此外有些博弈过程，在一个短时期内短兵相接，进攻和防御的战术变化剧烈，这些情况如何在搜索策略中加以考虑。还有基于极小极大过程的一些方法都设想对手总是走的最优走步，即我方总应考虑最坏的情况，实际上再好的选手也会有失误，如何利用失误加强攻势，也值得考虑。再一点就是选手的棋风问题。总之要真正解决具体的博弈搜索技术，有许多更深入的问题需要作进一步的研究和探讨。

习题

2.1 数字重写问题的变换规则如下：

6→3，3　　4→3，1

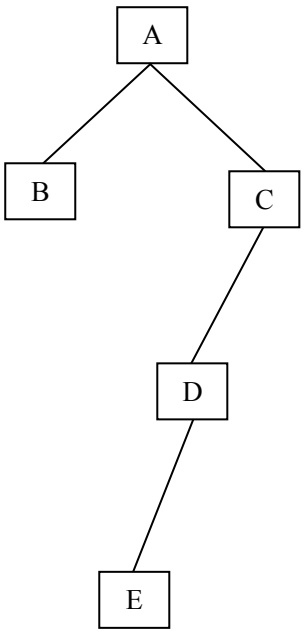
6→4，2　　3→2，1

4→2，2　　2→1，1

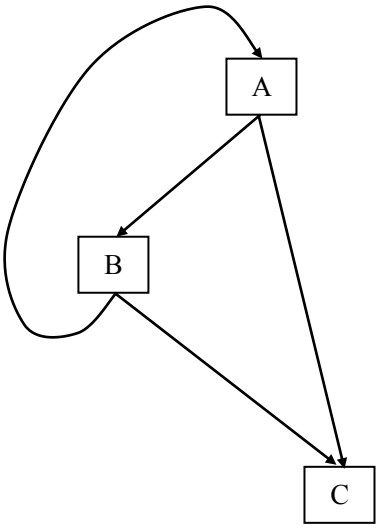
问如何用这些规则把数字 6 变换成一个由若干个 1 组成的数字串。试用算法 AO* 进行求解，并给出搜索图。求解时设 k-连接符的耗散值是 k 个单位，h 函数值规定为：

h（1）＝0，h（n）＝n（n≠1）。

2.2 AO*算法中，第 7 步从 S 中选一个节点，要求其子孙不在 S 中出现，讨论应如何实现对 S 的控制使得能有效地选出这个节点。如下图所示，若 E 的耗散值发生变化时，所提出的对 S 的处理方法应能正确工作。



题 2.2 的图



题 2.3 的图

2.3 如何修改 AO*算法使之能处理出现回路的情况。如上图所示，若节点 C 的耗散值发生变化时，所修改的算法能正确处理这种情况。

2.4 对 3×3 的一字棋，设用+1 和-1 分别表示两选手棋子的标记，用 0 表示空格，试给出一字棋产生式系统的描述。

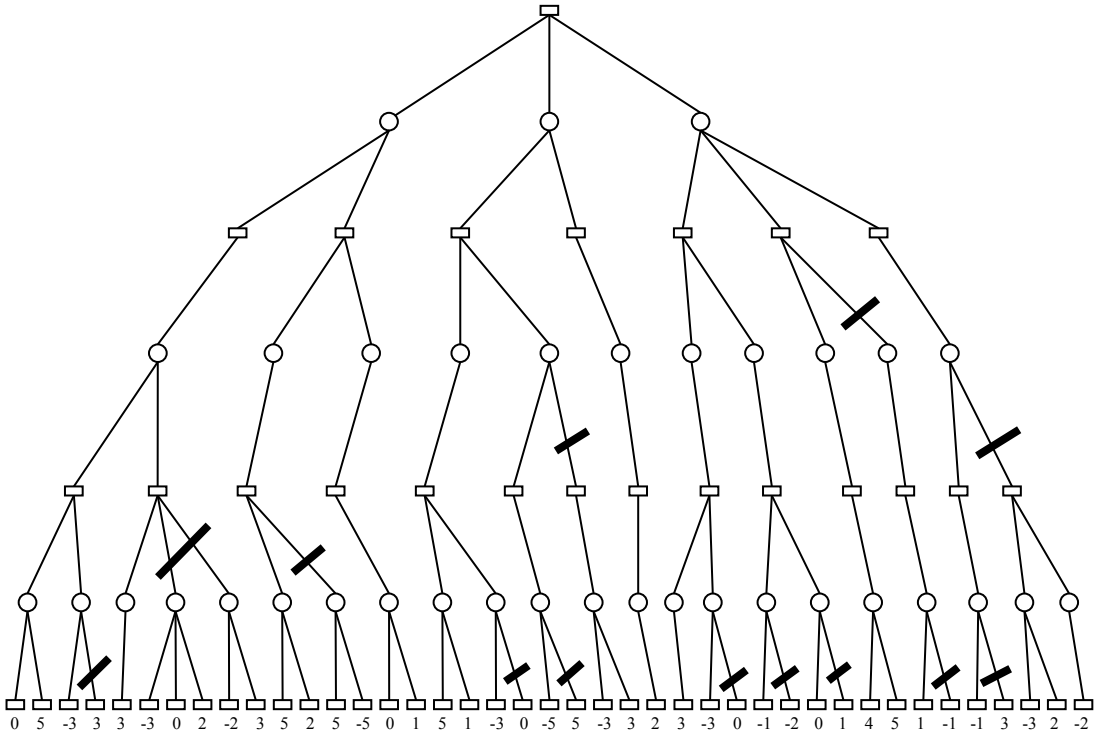
2.5 余一棋的弈法如下：两棋手可以从 5 个钱币堆中轮流拿走一个、两个或三个钱币，拣起最后一个钱币者算输。试通过博弈证明，后走的选手必胜，并给出一个简单的特征标记来表示取胜策略。

2.6 对题 2.6 图所示的博弈树，以优先生成左边节点顺序来进行 α - β 搜索，试在博弈树上给出何处发生剪枝的标记，并标明属于 α 剪枝还是 β 剪枝。

2.7 写一个 α - β 搜索的算法。

2.8 用一个 9 维向量 C 来表示一字棋棋盘的格局，其分量根据相应格内的×，空或○的标记分别用+1，0，或-1 来表示。试规定另一个 9 维向量 W，使得点积 C • W

可作为 MAX 选手（棋子标记为×）估计非终端位置的一个有效的评价函数。用这个评价函数来完成几步极小-极大搜索，并分析该评价函数的效果。



题 2.6 的图