博弈树搜索算法概述①

Overview of Game Tree Search Algorithm

岳金朋　冯　速　(北京师范大学 信息科学与技术学院 北京 100875)

摘　要： 机器博弈作为人工智能研究的重要分支，可研究的内容极为广泛。介绍现在计算机博弈中主流的搜索算法，并将它们有机的结合起来，给出一个搜索的主体框架，为博弈树研究者提供启发和参考。

关键词： 博弈树 极大极小算法 α－β剪枝 置换表

诸如下棋、打牌等类型的竞争性智能活动，称为博弈。最简单的一种是“二人零和、全信息、非偶然”博弈。博弈的实例有中国象棋、五子棋、国际象棋和围棋等。要提高计算机的下棋水平，就要有效地把多种搜索算法组合起来，进而改进博弈树的搜索效率来找到一步好棋。

1　博弈树

设博弈的双方中一方为甲方，另一方为乙方。在博弈过程中，某一方当前有多个行动方案可供选择时，他总是挑选对自己最为有利而对对方最为不利的行动方案。此时，如果我们站在甲方的立场上，则可供甲方选择的若干行动方案之间是“或”的关系，因为这时主动权在甲方手里，他可以选择这些行动方案中的任何一个行动方案。但是，若乙方也有若干个可供选择的行动方案，则对甲方来说这些行动方案之间是“与”的关系，因为乙方走棋时主动权在乙方手中，这些可供选择的行动方案中的任何一个都可能被乙方选中，甲方必须考虑对自己最不利的情况的发生。把上述博弈过程用图表示出来，得到的是一棵“与/或”树，参见图 1。这里要特别指出，“与/或”树始终是站在某一方(例如甲方)的立场上得出的，对于不同的走棋方“与/或”树不同。我们称这棵描述博弈过程的与/或树为博弈树(game tree)。其中，方形节点代表轮到甲方走棋的棋局，圆形节点代表轮到乙方走棋的棋局。

2　极大极小算法

极大极小算法(minimax algorithm)[1]是为博弈中的一方(例如甲方)寻找一个最优行动方案的方法。为了找到当前的最优行动方案，需要对各个行动方案可能产生的后果进行比较。具体地说，就是要考虑每一方案实施后对方可能采取的所有行动，并计算可能的分值。为计算分值，需要定义一个估值函数，用来估算当前博弈树叶节点的分值。当叶节点的估值计算出来后，再推算出父节点的分值。推算的方法是，对于“或”节点，选其子节点中一个最大的分值作为父节点的分值，这是为了使自己在可供选择的方案中选出一个对自己最有利的方案；对于“与”节点，选其子节点中一个最小的分值作为父节点的分值，即考虑最坏的情况。如果一个行动方案获得较大的倒推值，则它就是当前最好的行动方案。图 1 给出了计算倒推值的实例。为了表述方便，我们将取极大值的一方称为 Max 方，另一方称为 Min 方。

在博弈过程中，每一个格局可供选择的方案可以很多，因此会生成十分庞大的博弈树，据统计，中国象棋完整的博弈树约有 10^{150} 个节点。因此，可行的办法是生成一定深度的博弈树，然后用极大极小算法找出当前最好的行动方案。

3　α－β剪枝

3.1　α－β剪枝简介

用极大极小算法求倒推值的过程中，存在着两种明显的冗余现象[2]。第一种现象是极大值冗余，如图

① 基金项目:国家自然科学基金(60273015)
收稿时间:2008-11-23

© 中国科学院软件研究所 http://www.c-s-a.org.cn

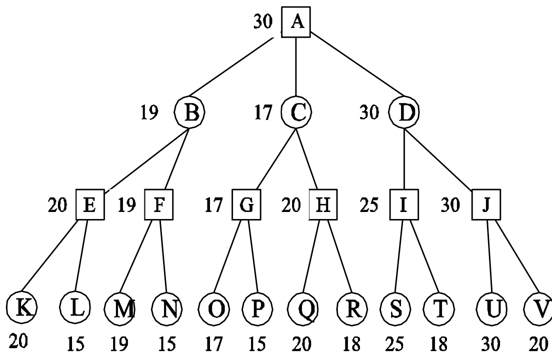
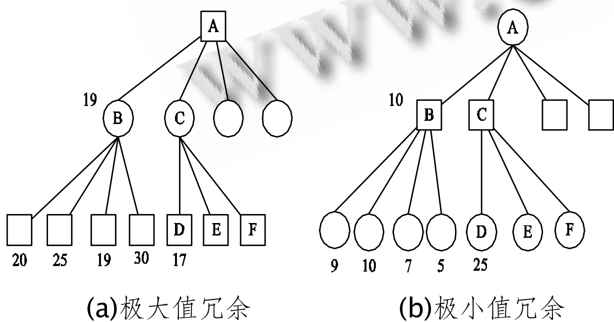


图 1　博弈树

2(a)所示的博弈树，我们要用极大极小值算法找出极大值节点 A 的最佳着法。先对 A 的第一个子节点 B 进行搜索，得出 B 的各个子节点的评价分别是 20、25、19 和 30，那么 B 的值为 19。节点 A 的值应是节点 B 和节点 C 的值中之较大者。现在已知节点 B 的值大于节点 D 的值。由于节点 C 的值应是它的诸子节点的值中之极小者，此极小值一定小于等于节点 D 的值，因此也一定小于节点 B 的值，这表明继续搜索节点 C 的其它诸子节点 E，F，…已没有意义，它们不能做任何贡献，于是可以把以节点 C 为根的子树全部剪去。这种优化称为 α 剪枝(alpha pruning)。图 2(b)是与极大值冗余对偶的现象，称为极小值冗余。节点 A 的值应是节点 B 和节点 C 的值中之较小者。现在已知节点 B 的值小于节点 D 的值。由于节点 C 的值应是它的诸子节点的值中之极大者，此极大值一定大于等于节点 D 的值，因此也一定大于节点 B 的值，这表明继续搜索节点 C 的其它诸子节点已没有意义，可以把以节点 C 为根的子树全部剪去，这种优化称为 β 剪枝(beta pruning)。



(a)极大值冗余　　(b)极小值冗余

图 2　α－β 剪枝示意图

3.2 窗口形式的 α－β 剪枝

在搜索过程中，Max 方节点的当前最优值被称为 α 值，Min 方节点的当前最优值被称为 β 值。在搜索开始时，α 值为－∞，β 值为＋∞，在搜索过程中，Max 节点使 α 值递增，Min 节点则使 β 值递减，两者构成一个区间[α，β]，这个区间被称为窗口[3-4]。窗口的大小表示当前节点需要搜索的子节点的价值取值范围，向下搜索的过程就是缩小窗口的过程，最终的最优值将落在这个窗口中。一旦 Max 节点得到其子节点的返回值大于 β 值或 Min 节点得到其子节点的返回值小于 α 值，则发生剪枝。

窗口形式的 α－β 剪枝可以进行更加充分的裁剪，它不仅可进行图 2 所示的浅层剪枝，还能将剪枝延伸到若干层之后。例如在图 3 所示的搜索树中，节点 F 的值为 12，而 G、H 和 I 的值都大于 12，因此节点 B 的评价就是 12。现在我们来搜索节点 C，在下面两层我们找到了使节点 N 的值变为 10 的着法，那么 N 返回到 J 的是 10 或更小的值。如果这个值也能从 J 返回到 C，那么我们可以在节点 C 上作裁剪，因为它有更好的兄弟节点 B。因此在这种情况下，继续找 N 的子节点毫无意义，所以我们可将以 N 为根的子树剪掉。

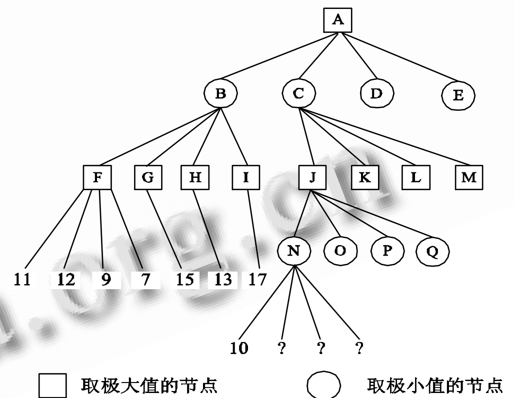


图 3　α－β 深剪枝示意图

通常所说的 α－β 剪枝就是指这种窗口形式的 α－β 剪枝，具体算法如下所示：

1)入口参数为(α，β，d)，[α，β]是初始窗口，d 是搜索深度，函数名为 AlphaBeta；

2)若 d＝0，则取静态估计值返回；

3)产生所有合理着法 mi，1≤i≤n；

4)Best＝－∞；

5)i＝1；

6)执行着法 m_i；

7)Score＝－AlphaBeta (－β，－α，d－1)；

© 中国科学院软件研究所 http://www.c-s-a.org.cn

8)撤销着法 m_i

9)若 Score＞Best，则 Best＝Score，否则，转 12);

10)若 Best＞ α，则α＝Best;

11)若 Best≥β，则取 Best 值返回;

12)i＝i＋1;

13)若 i≤n，则转 6);

14)取 Best 值返回;

在α－β剪枝中，搜索一个节点通常会出现三种情况[5]：高出边界型(fail high)，由于发生了剪枝，只知道返回评分至少是β，但不知道具体值；低出边界型(fail low)，由于没找到较好的着法，只知道返回评分最多是α，但不知道具体值；精确型(exact)，即返回评分在α和β之间。

3.3 α－β剪枝效率分析

如果我们将搜索树平均分枝因子数记作 b，搜索深度记作 d，那么采用极大极小算法搜索的节点数为：

$$1+b+b^2+b^3+...+b^d = b^d(1-1/b^d)/(1-1/b) \approx b^d \quad (1)$$

1975 年，Knuth 和 Moore 证明了，在节点排列最理想的情形下，使用α－β剪枝生成的节点数目为[6]：

$$\begin{aligned} N_d &= 2b^{d/2}-1 \quad &(d\text{为偶数}) \\ N_d &= b^{(d+1)/2}+b^{(d-1)/2}-1 \quad &(d\text{为奇数}) \end{aligned} \quad (2)$$

这个数值大约是极大极小算法搜索节点数的平方根的两倍。那么根据公式 $b^{d/2}=(b^{1/2})^d$，我们得出：如果着法顺序的排列最为理想，那么在同样的情况下，我们可以搜索原来深度(即不用α－β剪枝的深度)的两倍。由于α－β剪枝与节点的排列顺序高度相关，寻找有效手段将候选着法排列调整为剪枝效率更高的顺序就显得尤为重要了。

4 置换表

在博弈树中，不少节点之间虽然经过不同的路径到达，但其状态是完全一致的。通过建立置换表(Transposition Table)[7,8]，保存已搜索节点的信息，那么再次遇到相同状态的节点时便可套用之前的搜索结果，避免重复搜索。

4.1 Zobrist 哈希方法

为了快速检测当前节点是否已经搜索过，我们一般对棋盘状态进行编码，然后利用哈希表的方式进行查找，这里采用的编码方式是 Zobrist 哈希方法[7]：在搜索之前，生成大随机数数组 Zobrist [棋子类型][棋盘坐标]与代表走棋方的随机数 S。当前棋盘的哈希值便是棋盘上所有存在棋子的坐标对应的 Zobrist 值的异或之和。这样产生移动后不需要重新计算棋盘的哈希值，只需将当前棋盘的哈希值先与移动棋子的原坐标对应的 Zobrist 值做异或计算,然后与移动棋子的新坐标对应的 Zobrist 值做异或计算，如果产生吃子的话，需要再与被吃子对应坐标的 Zobrist 值做异或计算，最后与 S 做异或计算，表示走棋方的改变。这是根据两次异或同一个数结果保持不变的原理，避免重新计算整个棋盘的哈希值，并且位的异或在计算机内部运算速度较快。我们要用上述方法将每个局面映射到一个 32 位的索引值和一个 64 位的校验值，然后用这个 32 位索引值对置换表大小取模，找到此局面在置换表中的位置。然而，由于冲突的存在，我们必须验证此位置中存储的棋局与当前局面是否相同，而这个 64 位的校验值就可作为标签来区分每个局面。虽然 64 位的校验值理论上仍然可能存在冲突，但实战中这种情况非常罕见，因此可以忽略[7]。

4.2 置换表替换策略

我们利用 Zobrist 哈希方法让每一个局面在置换表中对应唯一的位置，但并不保证置换表中每一个位置对应唯一的局面，这样虽然将每一个局面都对应到置换表中，但却经常发生两个不同局面映射到同一位置而发生冲突的问题。如果向置换表中存储节点时发生了冲突，是否应该覆盖原来的项？对于这个问题有以下解决方案：(1)始终替换的方式[9]，即简单地覆盖已经存在的值。(2)同样深度或更深时替换的方式[9]，除非新局面的深度大于或等于哈希表中已有的值，否则保留已经存在的节点。(3)双层存储方式[10]，如果待置入节点的深度大于等于置换表第 1 层节点存储的深度，将当前第 1 层存储内容移至第 2 层，第 1 层存储写入待置入节点信息，否则，将待置入节点信息写入第 2 层中存储。(4)多层存储方式，一个 m 层的置换表有 n 个哈希位置，每个哈希位置记录 m 个局面，那么整个置换表可记录 m×n 个局面信息。记录一个局面时，找到它对应的哈希位置中的 m 个局面，不考虑该局面本身的搜索深度，而直接覆盖深度最小的那个局面。

4.3 置换表与α－β剪枝的结合

如前所述，在α－β剪枝中，一个节点会出现 fail high、fail low 和 exact 三种情况之一。一般只有 exact 型才可作为当前节点的准确值存入置换表中，但 fail high、fail low 所对应的边界值仍可帮助我们

© 中国科学院软件研究所 http://www.c-s-a.org.cn

作进一步的剪枝，因此在置换表中不仅要保存最优着法、分值、深度和当前局面的校验值，还需要设置一个标志域来说明节点的类型，例如，分值域中保存了 16，并且在标志域中保存了“exact”，说明节点的值是准确值；如果在标志域中保存了“fail low”，那么节点的值最多是 16；如果保存了“fail high”，这个值就至少是 16。在对某一局面进行搜索时，如果该局面已保存在置换表中，并且其深度域的值大于当前要搜索的深度，就可认为该置换表项中记录的分值是准确的，此时如果标志域中保存了“exact”，便可直接返回该分值而代替搜索，如果标志域为“fail high”而分值域保存的分值大于当前搜索的窗口上界 β 或标志域为“fail low”而分值域保存的分值小于当前搜索的窗口下界 α，便可直接剪枝。

置换表的使用是一种空间换取时间的思想，如果在置换表中能直接得到结果的话，则可以避免该节点以及该节点为根的子树的搜索，从而减少搜索时间。同时如果在置换表中能查找到当前节点的信息，并且存储深度比当前节点将要搜索的深度大，那么实际上是增加了当前子树的搜索深度，即增加了结果的准确性。正是因为置换表的诸多优点，所以置换表已成为博弈树搜索中广泛采用的技术。

5 选择性搜索

在人类下棋的时候，往往只选择几个认为较好的着法进行思考和深层次计算，而跳过一些较差的着法，这就涉及到如何延伸较好的着法以及忽略较差着法的思考方式。在程序中，需要对着法进行估算，当该着法达到某种标准时，对其进行剪枝，同时，对一些不稳定或容易产生分值起伏的着法进行延伸。

5.1 选择性裁剪

选择性裁剪是指不会影响结果正确性的剪枝。在实际应用中主要包括：(1)重复裁剪，主要指如果当前路径上出现重复局面就不再搜索下去；(2)和棋裁剪，主要指如果双方都没有明显可以杀死对方的子力就不再搜索下去并返回和棋分值；(3)如 4.3 节所述的置换裁剪。

5.2 选择性延伸

一般来说，考虑的深度越深，犯错误的概率就越小。所以可以考虑选择性延伸，有选择地增加感兴趣节点的搜索深度，提高该节点评价的准确度。当然，不能过度地对节点进行延伸，因为这样往往会造成搜索树过于庞大，以至于降低了搜索的效率。

选择性延伸通常运用在强制着法上，强制着法的界定在各个程序中有所不同，但主要有以下几种判断方法：(1)将军，此时必须应将，属于强制着法；(2)单一着法，走子的一方只有一种合理着法时，属于强制着法；(3)杀棋威胁，当一方不走子时就会被对方在几步内杀掉，那么解除杀棋威胁也属于强制着法，这种判断比较困难，通常利用下面所介绍的“空着搜索”来做判断。(4)兑子着法，大多数情况下将兑子着法也判断为强制着法。

一般极大极小方法用固定的深度进行搜索，每一步棋都搜索到一个固定的深度，这个深度被称为“水平线”。对于水平线以内发生的威胁，这个方法非常有效，但是它不能检查到水平线以后的威胁，这样就产生了水平线效应[11]。为了尽量减少水平线效应，我们还要在叶节点的局面变化剧烈时，继续向下搜索一直到局面相对平静为止(在中国象棋中，一般把由吃子走法所产生的局面和将军的局面看成是变化剧烈的，其余的局面则认为是相对静止的)，这样可以对一些比较混乱的局面做更精确的估计。这种延伸称为静止期搜索(quiescence search)。

在中国象棋程序中使用的延伸主要包括：将军延伸，兑子延伸。对于被将军的节点，一般将其搜索深度增加 1 层，这是因为一方面将军延伸能比较显著的提升棋力，另一方面可以解将的着法并不多；对于发生兑子的节点，一般将其搜索深度增加 3/4 或 2/3，这样的话我们可以接收小于 1 层的增量，当这些小于 1 层的增量累积起来，超过层数的整数表示值，便可进行额外的延伸，这样的做法是因为一方面吃子延伸在提升棋力方面不如将军延伸重要，另一方面吃子延伸会比较明显的降低搜索速度，况且，静止期搜索已经隐含了对叶子节点的吃子延伸。

5.3 空着裁剪

空着裁剪(Null-Move Pruning)[12]基于这样的思路：如果本方不走棋，而让对方连续走棋时，在减少深度的浅层搜索下也能使分值超过 β 的话，则进行剪枝。空着搜索实现较为简单且对速度的提升十分明显，现已被几乎所有博弈程序(除一些空着搜索没有意义的棋类)所采用。

空着裁剪的思想也是在适当时机调整搜索层数，但是它是通过相反的方式来表现的，即不是在复杂的局面上做延伸，而是在简单的局面上使用减少深度的搜索。

© 中国科学院软件研究所 http://www.c-s-a.org.cn

深度减少因子记为 R，可作为一个变量出现，如果对本方而言搜索深度为 D，空着搜索就是以 D－R－1 为深度搜索对手的着法。在应用中，应根据不同情况来对 R 值经行调整，这样的做法，称为“适应性空着裁剪”(Adaptive Null-Move Pruning)[13]，其内容可以概括为：深度小于或等于 6 时，用 R＝2 的空着裁剪；深度大于 8 时，用 R＝3；深度是 6 或 7 时，如果每方强子数都大于或等于 3 个，则用 R＝3，否则用 R＝2。该方法的依据是节点的搜索深度和强子的数目。

值得注意的是，空着裁剪在某些棋局上是不能正常运行的，这是因为我们假定走一步棋会比不走棋有更高的分值，这个假定在很多典型局面上并不成立，这些局面称为无等着局面(Zugzwang)。无等着局面指的是不走棋反而局势会更好些。空着裁剪应用到这些局面上会产生副作用，因此提出了带验证的空着裁剪(Verified Null-Move Pruning)[14]，其内容可以概括为：用 R＝3 的空着裁剪进行搜索；如果高出边界，那么做浅一层的搜索； 做浅一层的搜索时，子节点用 R＝3 的不带验证的空着裁剪；如果浅一层的搜索高出边界，那么带验证的空着裁剪是成功的，否则必须重新做完全的搜索。

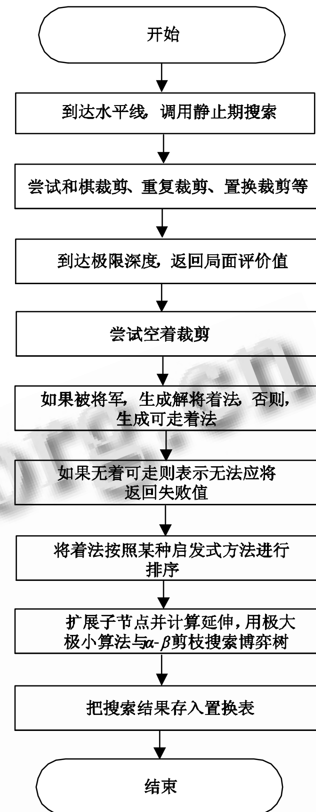


图 4 搜索主体框架

6 结论

以上对博弈树搜索算法中的关键技术做了大致的介绍。搜索算法是象棋程序的核心算法，在众多搜索算法中，如何选择适合自己的算法，并有效地将多种优化手段有机地结合在一起，是决定搜索效率的关键。图 4 给出了一种常用的博弈树搜索框架。

参考文献

1 Shannon CE. Programming a computer for playing chess. Philosophical Magazine, 1950,41(7):256－275.

2 王小春. PC 游戏编程(人机博弈).重庆:重庆大学出版社, 2002:102－103.

3 陆汝钤.人工智能(上).北京:科学出版社, 1995:390－392.

4 徐心和,王骄.中国象棋计算机博弈关键技术分析.小型微型计算机系统, 2006,27(6):961－969.

5 Eppstein D. Hashing and Move Ordering. 1997/4. http://www.ics.uci.edu/~eppstein/180a/970424.html.

6 Knuth DE, Moore RW. An analysis of Alpha-Beta pruning. Artificial Intelligence, 1975,6(4):293－326.

7 Zobrist A. A new hashing method with application for game playing. ICCA Journal, 1990,13(2):69－73.

8 Moreland B. Transposition table. 2004/3.https://chessprogramming.wikispaces.com/Transposition+Table.

9 Breuker DM, Uiterwijk JWHM, Herik HJ van den. Replacement schemes for transposition tables. ICCA Journal, 1994,17(4):183－193.

10 Breuker DM, Uiterwijk JWHM, van den Herik HJ. Replacement schemes and two-level tables. ICCA Journal, 1994,19(3):175－180.

11 Eppstein D. Which nodes to search? Full-width vs. selective search. 1999/2. http://www.ics.uci.edu/~eppstein/180a/990204.html

12 Donninger C. Selective search heuristics for obtuse chess programs. ICCA Journal, 1993,16(3):137－143.

13 Heinz EA. Adaptive Null-Move Pruning. ICCA Journal, 1999,22 (3):123－132.

14 Tabibi OD , Netanyahu NS. Verified Null-Move Pruning. ICCA Journal, 2002,25(3):153－161.

© 中国科学院软件研究所 http://www.c-s-a.org.cn