博弈算法在黑白棋中的应用

杜秀全，程家兴
（安徽大学 计算智能与信号处理教育部重点实验室，安徽 合肥 230039）

摘 要：计算机博弈是一种对策性游戏，是人工智能的主要研究领域之一，它涉及人工智能中的搜索方法、推理技术和决策规划等。目前广泛研究的是确定的、二人、零和、完备信息的博弈搜索。文中通过一个黑白棋程序的设计，将生成的博弈树节点的估值过程和对博弈树搜索过程相结合，采用传统的 Alpha－Beta 剪枝和极大－极小原则方法给出了博弈程序设计的核心内容：包括博弈树搜索和估值函数两个方面，提出了对原算法的一种改进，该算法提高了搜索速度。实验结果验证了算法的有效性。

关键词：博弈树；黑白棋；估值函数；人工智能
中图分类号：TP311.1　　**文献标识码**：A　　**文章编号**：1673－629X(2007)01－0216－03

Game－Playing Algorithm in Black and White Chess Application

DU Xiu-quan，CHENG Jia-xing
（Ministry of Education，Key Laboratory of Intelligent Computing and Signal Processing，Anhui University，Hefei 230039，China）

Abstract：The computer game－playing is a game of countermeasure and one of the major artificial intelligence research areas. It involves reasoning，decision－making and planning etc. At the present，widely study the simple basic game－playing that is determinate，two person，zero，complete information game－playing search. Through a black and white chess procedure design，introduced the game－playing programming core content by the process of estimating the production tree of game－playing point value and the tree of game－playing search，used traditional Alpha－Beta pruning and the principle of max－min method to introduce the gambling programming core content，including the tree of game－playing search and valuation function，put forward the algorithm improvement method which increase searching speed，the experimental result validated its effect.

Key words：tree of game－playing；black and white chess；evaluation function；artificial intelligence

0 引 言

人工智能是近年来很活跃的研究领域之一，计算机博弈是人工智能研究的重要分支[1]。黑白棋在国外是一种深受大众广泛喜爱的游戏，其规则简单，变化多端，非常富有趣味性和消遣性。黑白棋程序设计是用编程的方法教会电脑下棋，使之可以与对手对抗。这里采用 Visual C＋＋ 6.0 设计博弈黑白棋程序，应用了 Alpha－Beta 剪枝和极大－极小原则进行搜索最佳位置。

1 博弈算法

在博弈(Game Playing)中只有“敌、我”二方，并且他们的利益是完全对立的，其赢得函数和为零：$C1+C2=0$。其中：$C1$ 为我方赢得(利益)；$C2$ 为敌方赢得(利益)。

因此只有三种情况出现：

(1) 我方胜 $C1>0$，则 $C2=-C1<0$；

(2) 敌方胜 $C2>0$，则 $C1=-C2<0$；

(3) 平　局 $C1=0$，则 $C2=0$。

在程序设计过程中，采取“极大－极小(Min－Max)分析法”，即在博弈树的搜索过程中，估值函数 $E(x)$ 反映双方赢得的大小，从我方出发，可取估值函数[2]为：

$E(x)=C1(x)$，$C1>0$ 得分，$C1<0$ 失分。

双方都根据“极大－极小”的原则，在按照博弈规则生成的博弈树中，选取最佳步[3]，即：

我方最佳棋步：取 max[E(x)]，在我方得分 C1 最大，扩展“或”节点；

敌方最佳棋步：取 min[E(x)]，在我方得分 C1 最小，扩展“与”节点。

2 极大－极小原则

例：初始棋局如图 1 所示。

由极大－极小原则可以形成如图 2 所示的搜索树。

结果应下 $E3$ 或 $C3$。

收稿日期：2006－04－07

作者简介：杜秀全(1982－)，男，安徽全椒人，硕士研究生，研究方向为智能计算理论与应用、优化方法；程家兴，教授，博导，研究方向为智能计算与优化方法等。

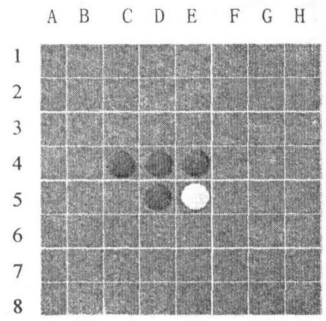

图 1　初始棋局

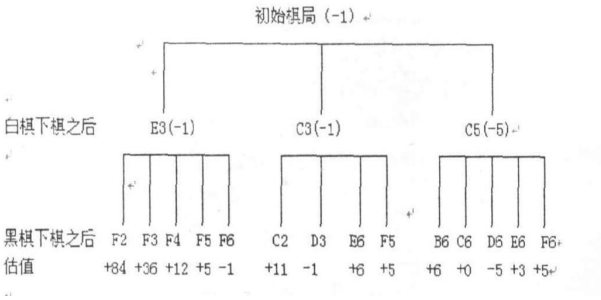

图 2　博弈搜索树

3　Alpha－Beta 剪枝

对于一般的极大极小搜索,即使每一步只有很少的下法,搜索位置都会呈指数级增长,但可以证明,程序不要考虑所有的位置而找到最佳位置[3],这就是 Alpha－Beta 剪枝法(如图 3 所示),其基本思想是[3]:

(1) 若"或"节点的 α 值,不能使其父节点的 β 值降低,则该"或" 节点以下的分支树不要再搜索,称为 β 剪枝。

(2) 若"与"节点的 β 值,不能使其父节点的 α 值升高,则该"与" 节点以下的分支树不要再搜索,称为 α 剪枝。

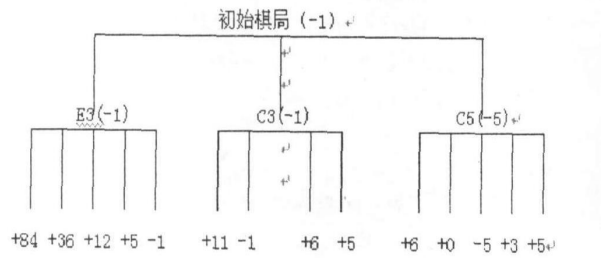

图 3　Alpha－Beta 剪枝示意图

假设先搜索 $E^3-F^2\ F^3\ F^4\ F^5\ F^6$,然后是 $C^3-C^2\ D^3\ E^6\ F^5$、$C^5-B^6\ C^6\ D^6\ E^6\ F^6$,即按从左到右的顺序进行深度优先搜索。则搜索到 D^3 分枝之后,就不用搜索 E^6 和 F^5,因为如果接下来的值比 D^3 大,就不会把值给 C^3,如果比 D^3 小,把值给 C^3 后,也不会给根节点,因为根节点取最大值。同样,搜索 D^6 之后,就不再搜索 E^6 和 F^6 了 。

Alpha－Beta 剪枝算法在黑白棋中的应用:

```
int AlphaBeta(nPly, nAlpha, nBeta)
{
  if (game over)
    return Evaluation;//胜负已分,返回估值
  if (nPly == 0)
    return Evaluation;//叶子节点返回估值
  if (Is Min Node) //此句用于判断当前节点是何种节点
    {//是取极小值的节点
      for (each possible move m) //对每一可能的走法 m
        {
          make move m;//生成新节点
          score =AlphaBeta(nPly－1, nAlpha, nBeta);//递归搜索子节点
          unmake move m;//撤销搜索过的节点
          if (score < nBeta)
            {
              nBeta=score;//取极小值
              if (nAlpha > = nBeta)
                return nAlpha;//剪枝,抛弃后继节点
                }
              }
            }
          return nBeta;//返回最小值
        }
      else
        {//取极大值的节点
          for (each possible move m) //对每一可能的走法 m
            {
              make move m;//生成新节点
              score =AlphaBeta(nPly－1, nAlpha, nBeta);//递归搜索子节点
              unmake move;//撤销搜索过的节点
              if (score > nAlpha)
                {
                  nAlpha=score;//取极大值
                  if (nAlpha > = nBeta)
                    return nBeta ;//剪枝,抛弃后继节点
                    }
                  }
                return nAlpha;//返回最大值
              }
        }
```

4　算法改进

当搜索深度稍大时,就会出现速度很慢的现象,于是做了以下改进:

(1)改变搜索顺序会提高剪枝算法的效率[4]。

比如在刚才的搜索中,如果搜索顺序为 $D^6-B^6-C^6-E^6-F^6$,则搜索到 D^6 后,后面的 4 个节点都被剪掉了,但如果是 $B^6-C^6-E^6-D^6-F^6$,则搜索到 D^6 后,后面只有一个节点被剪掉,而且还多搜索了 3 个节点。怎么确定搜索顺序呢?可以有两种方法:

① 如果对手下了 G^2,则就下 H^1,看角是否可占。

②执行浅搜索。

(2)当算法给出所选的走步后,不要马上停止搜索,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[5]。

5 估值函数

这是这个程序中最重要的部分,如果这个模块太弱,那么算法再好也没有用。估值函数有很多种,主要有:

a)棋格表。

不同的棋格有不同的值,角的值大而角旁边的格子值小,每个格子有三种可能性值:黑棋、白棋和空,它比较容易实现。

b)基于行动力的估值。

c)估值合并[1]。

一般采用线性合并。设 $a1, a2, a3, a4$ 是基于不同参数得到的估值,则合并后的估值 $s = p1 * a1 + p2 * a2 + p3 * a3 + p4 * a4$,其中 $p1, p2, p3, p4$ 为常数。

6 黑白棋程序实验与分析

本程序中用到的类如下:

```
Class CBwfind:Cobject
{
  Public://保存棋局
        Void SaveBoard(int tb[][8]);
        Int outcome;
        Int bak;
    //记录 64 步的棋局,以便悔棋
        Int save[64][8][8];
        Bool preview;
        Int stepx,stepy;
    //搜索
        Bool search(int tb[][8],int);
        Int depth;
        Int searchend;
        Int countnum();
        Int countwhite();
        Int countblack();
        Void select(int);
        Void newboard(int tb[][8],int,int,int);
        Bool decide(int,int,int);
        CBwfind();
        Int board[8][8];
        Virtual CBwfind();
        Int ComChoice,ManChoice;
Protected:
Private:
      //估价函数
        Void corner(int tb[][8],int I,int j,int &value);
        Void linesense(int tb[][8],int chessx,int chessy,int &value);
        Int mobility(int tb[][8],int flag);
        Void cornersense(int tb[][8],int &);
        Int valuefirst(int tb[][8]);
        Int findfirst(int tb[][8],int &,int &,int);
      //计数
        Int white(int tb[][8]);
        Int black(int tb[][8]);
        Int total(int tb[][8]);
        Bool judge(int tb[][8],int chessx,int chessy,int flag);
        Int findlast(int tb[][8],int &,int &,int);
}
```

通过实验(实验结果见表 1)可知:Alpha—Beta 剪枝和那些没有剪枝的程序相比走步要快的多,而添加辅助搜索的程序在智力和时间有效性方面明显要好于这些程序,智力和搜索 3 步相当。

表 1 实验结果

盘数	没有剪枝(搜索 2 步)	Alpha—Beta 剪枝(搜索 3 步)	Alpha—Beta 剪枝(搜索 2 步向前辅助搜索 1 步)
15	9	12	12
20	12	16	16
30	16	24	23

7 小 结

文中介绍了两种基本搜索方法,同时从博弈树搜索和估值函数优化两个方面,对上述博弈算法给出了改进的方法。在 Windows 操作系统下,用 VC++实现了这个人机对战的黑白棋程序。和国内许多只是采用规则或者只是采用简单递归而没有剪枝的那些程序相比,在智力上和时间有效性上都要优于这些程序。

参考文献:

[1] 王小春.PC 游戏编程(人机对弈)[M].重庆:重庆大学出版社,2002:123—128.

[2] 王小春.游戏编程(人机博弈)[M].重庆:重庆大学出版社,2002.

[3] 马少平,朱晓燕.人工智能[M].北京:清华大学出版社,2004:64—74.

[4] Pearl J.Heuristics—Intelligence Search Strategies for Computer Problem Solving[M].[s.l.]:Addision —Wesley Pub Co,1984.

[5] 蔡自兴.人工智能及其应用[M].北京:清华大学出版社,1999.