文章编号：2095-2163(2022)01-0128-04　　中图分类号：TP312　　文献标志码：A

基于改进 UCT 算法的国际跳棋博弈系统研究

张家铭，王静文，李　媛
（沈阳工业大学 理学院，沈阳 110870）

摘　要：国际跳棋的博弈系统中，UCT 算法是一个比较新颖的算法，其效果得到了广泛认可。但是 UCT 算法的缺点也同样明显，终局下的处理明显不如稳定性更强的 Alpha-Beta 算法。为避免 UCT 算法的不稳定性，本文采取限制搜索深度和增加扩展条件的方法对 UCT 算法进行改进，改进后的 UCT 算法胜率远超过改进前的 UCT 算法胜率，使得博弈水平得到了极大的提升。

关键词：国际跳棋；UCT 算法；Alpha-Beta 算法；UCT 算法的不稳定性

Draughts based on improve UCT algorithm

ZHANG Jiaming，WANG Jingwen，LI Yuan
（School of Science，Shenyang University of Technology，Shenyang 110870，China）

【Abstract】Against draughts game system，UCT algorithm is a relatively new algorithm，and its effect has been widely recognized. However，the shortcomings of UCT algorithm are also obvious，which is obviously inferior to Alpha-Beta algorithm with stronger stability in the final game，In order to avoid the instability of UCT algorithm，this paper adopts the method of limiting the search depth and increasing the extension conditions to improve UCT algorithm，and compares with the improved UCT algorithm，it is concluded that the improved UCT algorithm win rate is much higher than the improved UCT algorithm，which greatly improves the game level.

【Key words】draughts；UCT algorithm；Alpha-Beta algorithm；instability of UCT algorithm

0　引　言

随着计算机博弈项目的发展，随之而来的研究也变得越来越多元化，计算机战胜人类成为一个热门的话题。国际跳棋在计算机博弈方面是比较受欢迎的一个棋种。目前国际跳棋 64 格已经被解决，每一种局面都算出了最优解[1]。而国际跳棋 100 格却有着更加复杂的情况，本文通过对国际跳棋 100 格的研究，得出对棋盘的一种评估函数，并对 UCT 算法进行改进与测试，并且通过对 UCT 算法改进前后对比，得出改进 UCT 算法对局面的判断更为准确的结论。

1　国际跳棋简介

1.1　国际跳棋棋盘

国际跳棋是一种古老的游戏，棋盘为 10×10 黑白格相间的棋盘，在整个对弈的过程中，白色格子自始至终都是用不到的，棋盘放在对弈双方的中间，每个玩家的右下角应是白色格子，如图 1 所示。黑棋先手，然后双方轮流走动己方棋子。

棋子自始至终都是沿着对角线移动和跳吃，对弈的目标是将对方所有棋子吃掉或者形成一个局面迫使对方棋子无法移动。

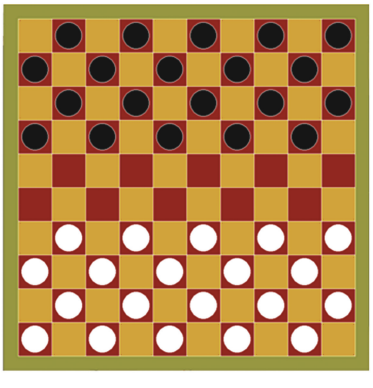

图 1　国际跳棋棋盘

Fig. 1　The board of Draughts

作者简介：张家铭(2001-)，男，本科生，主要研究方向：计算机博弈；王静文(1965-)，男，学士，工程师，主要研究方向：人工智能、信息安全；李　媛(1976-)，女，博士，教授，主要研究方向：人工智能和随机过程。
通讯作者：王静文　　Email：wangjingwen007@126.com

收稿日期：2021-10-15

1.2　国际跳棋下法规则

棋子可以进行跳吃下法，只要对角线方向邻近的黑格内有对方的棋子，并且再过去的黑格是空位，就可以跳过对方的棋子并将对方棋子吃掉；如果没有跳吃的下法，那就只能沿对角线方向前移一格；任何一个棋子到达了对方底线便立刻加冕，从此以后便加冕成“王”；跳吃可以由多次跳吃组成，如果具备连续跳吃的条件，则必须连续跳吃，除非不再具备跳吃的条件，才可以结束跳吃。未加冕的棋子只能向前移动，而在跳吃时可以向前、向后或前后组合。需要注意的是，只有停在对方底线的棋子才可以加冕成王，如果有一个棋子在跳吃过程中途经底线，但是最后没有落在底线，则该棋子无法加冕成王；王棋可以在对角线方向上移动任意多个空格，同样在跳吃的时候，王棋可以跳过对方棋子前后任意数量的空格。所以王棋比一般棋子要珍贵而强大，然而未加冕棋子是可以吃掉王棋的；跳吃的时候，在具有多种选择的情况下，必须选择吃子最多的下法。如果不止一个棋子或者不止一个路线可以跳吃对方同样最多的棋子，就可以自主选择哪个棋子或者向哪个方向行进[2]。

2　国际跳棋博弈系统

计算机博弈游戏直接决定棋力的关键因素为搜索算法与局面评估函数，现在比较传统的算法包括 Min-Max 搜索、负极大极小搜索、Alpha-Beta 搜索等等[3]。此类搜索算法均有一个关键是搜索的深度是固定的 d，当搜索深度达到 d 时，再依次计算估值进行回传，由于不同分支的搜索深度相同，计算量呈指数增加，极大的影响了搜索的速度。

目前比较新颖的算法为 UCT 算法，其特点是实现了对不同结点进行深度不同的搜索，并且搜索次数与搜索时间是可控的，若进行足够的模拟一般都可获得较为准确的结果。

2.1　国际跳棋的局面评估

在 Min-Max 算法以及 Alpha-Beta 等算法中，局面评估函数直接决定了对弈棋力的好坏。显然己方与对方的残留棋子之差与王棋子之差的权值应为最大。

当双方在前期局面对弈时，利用己方与敌方换子等操作来调整双方阵容，进行攻击或者防守；当双方在中局对弈时，往往可移动的棋子数量不多，于是需要适当的可移动棋子来使本方不至于陷入被动；当双方在残局对弈时，有时故意卖子来使对方被动的可能性是存在的，此时搜索深度需要进一步的提高。综上所述，本文的局面评估主要依靠如下几个方面：

（1）己方棋子数量 X_1，对方棋子数量 X_2；

（2）己方王棋数量 X_3，对方王棋数量 X_4；

（3）当己方棋子构成一列时，对方棋子不易对该列形成打击，于是己方构成列值数量 L_1，对方棋子构成列值数量 L_2；

（4）当棋子处于棋盘的中间位置时，更容易对另一方棋子造成致命性打击，并限制对方的棋子移动[4]，本方棋子在中间的棋子数为 Z_1，对方棋子构成的列数为 Z_2；

（5）当本方棋子接近对方底线时，该棋子升王的概率会大幅度增加，进而引入平衡因子来评估棋子接近对方底线的程度，己方棋子的平衡因子 B_1，对方棋子的平衡因子 B_2，对某一局面的价值评估为式(1)：

$$V = \omega_1(X_1 - X_2) + \omega_2(X_3 - X_4) + \omega_3(L_1 - L_2) + \omega_4(Z_1 - Z_2) + \omega_5(B_1 - B_2) + \cdots\cdots \quad (1)$$

其中，$\omega_i(i = 1, \cdots, n)$ 为每个因子的权重。

但是需要注意的是真正的局面评估函数往往不是条件越多效果越好的，评估因子的选取往往需要通过具体的局面具体的估值，在国际跳棋还有许多可以利用的估值[5]。

2.2　Alpha-Beta 算法

Alpha-Beta 算法是 Min-Max 算法的一种改进，Alpha-Beta 算法增加的为 Alpha-Beta 剪枝，即将某些搜索无意义的节点进行剪枝处理，大大的提高了搜索的效率。

Alpha-Beta 算法的伪码：

```
Int AlphaBeta(int depth,int alpha,int beta)
{
  如果 (depth == 0) return 局面评估();
  获取合法的下法;
  while(每一种下法)
  {
    执行走法;
    价值 = - AlphaBeta(depth - 1, - beta, - alpha);
    复原局面;
    如果(价值>=beta) return beta;
    如果(价值> alpha) alpha = val ;
  }
  return alpha;
```

}

Alpha-Beta 算法改进其中之一为迭代加深。迭代加深的是若给每一步都分配一个时间,当某一层搜索结束后,若时间仍然有剩余,则进行下一层的搜索,执行这种操作的好处是可以在给定的时间内进行尽可能深的搜索。

迭代加深的伪码:

for ($depth = 1;;depth++$)

{

价值= $AlphaBeta(depth$,负无穷,正无穷);

如果(超出给定时间) break;

}

2.3 UCT 算法

UCT 算法又名上限置信区间算法,该算法将蒙特卡洛树搜索(Monte-Carlo Tree Search)算法与 UCB 公式进行结合,是一种通过对大量节点的模拟,来获得最优解的算法[6]。该算法在搜索过程中,针对不同的分支的搜索深度可以不同,因此,对于某些节点,可以获得更深的搜索深度。

UCT 算法的计算公式(2)为[7]:

$$r_i = v_i + c \times \sqrt{\frac{2\ln(\sum_i T_i)}{T_i}} \tag{2}$$

其中, v_i 是节点 n_i (非叶子节点)为根节点的所有仿真结果的平均值,反映了该节点 n_i 的胜率; T_i 为节点 n_i 的访问次数,也是被树内选择策略选中的次数; c 是一个常数,用来调节深度优先与宽度之间的平衡。

UCT 算法的执行过程如图 2 所示,UCT 算法 4 个具体过程为:

(1) 选择:从根节点出发,依次对每一个根节点的孩子节点进行选择,选择 r_i 最大的节点,并从此节点开始,采用递归的方法选择,直到叶子节点;

(2) 扩展:当选择到叶子节点时,将所有合法的下法加到该搜索树中,并初始化 r 值和 T 值;

(3) 模拟:对上一步所有合法的下法,依次进行随机下棋,直到终局,如果最后本方胜利,则己方 $wins$ 为 1,若对方胜利,则己方 $wins$ 为 0;

(4) 反向传播:当叶子节点模拟得出新的 v 和 T 值时, UCT 算法将结果回传,更新搜索路径上的所有节点的 v 和 T 值,其计算公式(3)和公式(4)为:

$$T = \sum_i T_i \tag{3}$$

$$v = \frac{\sum_i v_i T_i}{T} \tag{4}$$

即父节点的访问次数 T 为所有孩子节点的访问次数 T 之和,将结果从叶子结点开始,依次上传到根节点为止。

需要注意的是如果胜率最高与访问量最高的移动不同时,UCT 算法往往是不稳定的,需要对 UCT 算法进行改进操作[8]。

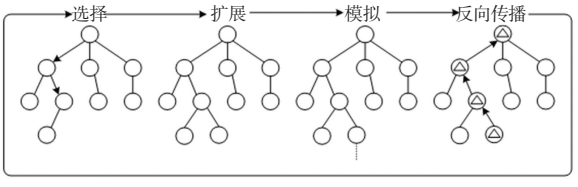

图 2 UCT 算法的基本过程

Fig. 2 UCT Basic Process

2.3.1 UCT 算法的改进

UCT 算法第一个缺点是无论在什么局面下都进行扩展操作,从而导致如果某一局面对本方无利,那么该算法同样会对该局面进行扩展研究;第二个缺点是当 UCT 算法进行无限制的扩展后,往后的局面可能会变得更加复杂,容易导致判断失误。为了避免这种情况的发生,故对 UCT 算法进行改进操作。

对于某一局面来说,当 UCT 算法对某一节点重复选择 M 次后,才对该节点进行扩展,且扩展后的最大深度不超过 N 层。

2.3.2 UCT 算法的不稳定性

通过不断测试的结果中发现, UCT 算法在接近终局时,当本方胜率过高时,UCT 算法的移动就会极具不稳定性,具体表现为会选择一些错误的下法[9]。本文所采取的方法是在接近终局时更换其他稳定性更高的算法。

3 算法比较

3.1 UCT 算法的调整

UCT 算法中 c 的取值不相同,对 UCT 算法的强度有很大的影响,这里取节点重复选择次数 M 为 20 次,最大搜索深度为 14 层,搜索时间为 15 s,由于 UCT 算法的不稳定性,前 38 步使用 UCT 算法,而后更换为 9 层 Alpha-Beta 算法,对方是 9 层 Alpha-Beta算法,采取的测试平台是 CPU:Intel i7-8700, 内存:DDr4 2666 MHz 8g, GPU:GTX 1050ti。

测试结果见表 1、表 2 和图 3。

表 1　UCT-AlphaBeta 对弈结果表

Tab. 1　Results of UCT-vs-AlphaBeta

UCT c 值	UCT 胜率	UCT 胜盘数	UCT 负盘数	平盘数
0.80	0.307	35	37	42
0.70	0.282	33	41	43
0.65	0.393	44	31	37
0.60	0.482	67	34	38
0.55	0.398	47	35	36
0.50	0.298	34	46	34
0.40	0.342	41	40	39

表 2　AlphaBeta-UCT 对弈结果表

Tab. 2　Results of AlphaBeta-vs-UCT

UCT c 值	UCT 胜率	UCT 胜盘数	UCT 负盘数	平盘数
0.80	0.394	52	41	39
0.70	0.356	57	47	56
0.65	0.373	44	24	50
0.60	0.358	58	44	60
0.55	0.410	50	27	45
0.50	0.356	58	40	65
0.40	0.385	47	26	49

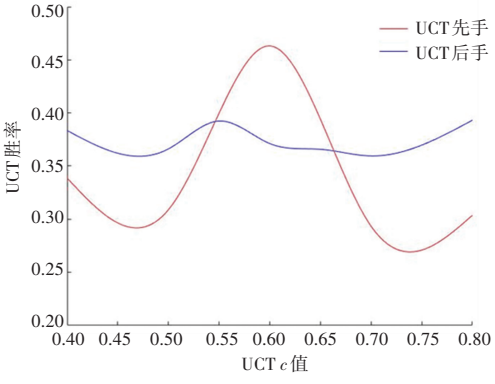

图 3　UCT 胜率变化

Fig. 3　UCT Win Rate change Table

由图 3 可以看出，选择了 7 个不同的 c 值进行对比，UCT 算法先手方 c 值在 0.6 处胜率取得极大值，UCT 算法后手方最佳的 c 值仍然在 0.6 附近。

3.2　UCT 算法与 Alpha-Beta 算法比较

通过原始的 15 s UCT 算法与 7 s 迭代加深算法进行对比，选取 c 值为 0.6，由于 UCT 算法的不稳定性，UCT 方前 43 步使用 UCT 算法，而后使用 10 层 AlphaBeta 算法，对方采取 10 层 AlphaBeta 算法，对比结果见表 3 和表 4。

表 3　UCT-AlphaBeta 对弈结果表

Tab. 3　Results of UCT-vs-AlphaBeta

对弈总数	UCT 胜率	UCT 胜盘数	UCT 负盘数	平盘数
630	0.165	104	362	164

表 4　AlphaBeta-UCT 对弈结果表

Tab. 4　Results of AlphaBeta-vs-UCT

对弈总数	UCT 胜率	UCT 胜盘数	UCT 负盘数	平盘数
859	0.081	70	645	144

3.3　改进 UCT 算法与 Alpha-Beta 算法比较

这里改进 UCT 算法 c 值选取 0.6，对某一节点重复选择次数 M 为 35 次，搜索最大深度 N 为 12 层。由于 UCT 算法的不稳定性，UCT 方前 43 步使用 UCT 算法，而后改为 10 层 AlphaBeta 算法，对方采取 10 层 AlphaBeta 算法。对比结果见表 5 和表 6。

表 5　改进 UCT-AlphaBeta 对弈结果表

Tab. 5　Results of Improve UCT-vs-AlphaBeta

对弈总数	UCT 胜率	UCT 胜盘数	UCT 负盘数	平盘数
560	0.350	196	193	171

表 6　AlphaBeta-改进 UCT 对弈结果表

Tab. 6　Results of AlphaBeta-vs-Improve UCT

对弈总数	UCT 胜率	UCT 胜盘数	UCT 负盘数	平盘数
757	0.395	299	234	224

4　结束语

通过 UCT 算法改进前与 UCT 算法改进后的对比可以发现，改进后的 UCT 算法胜率远远大于原始的 UCT 算法，并且在与 Alpha-Beta 算法对弈的前中期过程中，往往是改进 UCT 方占据优势，从而可以得出采取改进 UCT 算法的确可以使博弈水平得到提升。本文所选取的节点重复选择次数 M 和搜索最大深度 N，也许不是最佳的，可以尝试更改 M 和 N 的值来使得改进后的 UCT 算法进一步优化。

参考文献

[1] SCHAEFFER J, et al., Checkers Is Solved[J]. Science, 2007, 317(5844): 1518-1522.
[2] 杨周凤. 国际跳棋完备信息博弈关键技术研究与系统设计[D]. 沈阳:沈阳航空航天大学,2018.
[3] 吴岳. 一种高性能西洋跳棋引擎设计[J]. 电脑编程技巧与维护, 2014(2): 63-67,82.
[4] 安萌萌,李淑琴. 一种西洋跳棋评估算法[J]. 北京信息科技大学学报, 2016,32(2):85-88.
[5] 郑昌松. 基于西洋跳棋的博弈程序研究[J]. 哈尔滨理工大学学报, 2016(3):24-28.
[6] MAGNUSON M. Monte Carlo Tree Search and Its Applications [J]. Division of Science and Mathematics, 2015.
[7] STURTEVANT N R. An Analysis of UCT in Multi-player Games [C]// International Conference on Computers and Games. 2008.
[8] 张玉琪. 基于静态评估的计算机围棋 UCT 算法改进研究[D], 南昌:南昌航空大学, 2015.
[9] HUANG S, et al. MoHex 2.0: A Pattern-Based MCTS Hex Player [C]// International Conference on Computers and Games. 2013.