摘要

这是一个测试。

关键词：你好，我好，他好

Abstract

This is a test.

Keyword: Dog, Cat, Pig

目 录

第一章 绪论

1.1 课题的背景

1.2 中国象棋计算机博弈研究现状

1.3 本文主要工作及难点

1.4 本文的结构安排

第二章 基本模块

2.1 引言

2.2 局面数据结构

2.2.1 棋盘表示

2.2.2 着法生成

2.2.3 哈希函数及置换表

2.3 局面评价函数

2.3.1 引言

2.3.2 子力平衡评价

2.3.3 车的灵活性及马的阻碍

2.4 着法排序

2.4.1 引言

2.4.2 置换表启发

2.4.3 静态评价启发

2.4.4 动态启发

2.4.5 结论及测试结果

2.5 本章小结

第三章 搜索方法

3.1 引言

3.2 Alpha-Beta 算法

3.3 主要变例搜索

3.4 迭代加深

3.5 静态搜索

3.6 结果测试

3.7 本章小结

第四章 基于评价函数的动态博弈树
4.1 引言
4.2 建立动态博弈树
4.3 维护动态博弈树
4.4 测试结果并对比
4.5 本章小结
第五章 全文总结及展望
5.1 全文总结
5.2 后续工作展望
致谢
参考文献
附录
外文资料原文
外文资料译文

第一章 绪论

1.1 课题的背景

早在人类文明发展初期，人们就已经开始进行棋类博弈的游戏了。可以说，进行棋类博弈是人类智慧的一种体现。

在人工智能领域，机器博弈一直被认为是最具有挑战性的课题。让计算机拥有博弈的能力，与就意味着计算机拥有了一定的智能。因此，人工智能领域的学者可以在机器博弈这个平台上检验自己的研究成果；对机器博弈进行研究而产生出来的技术也已广泛应用于政治、经济、军事等领域中，并取得了大量引人瞩目的成果。也可以说机器博弈是人工智能的一块实验田。

早在计算机发展的初级阶段，香农（1950）以及图灵（1953）就已经提出了对国际象棋博弈策略及程序的描述。香农指出，计算机博弈的实质就是生成博弈树，并对博弈树进行搜索。他还提出了两种博弈树搜索策略：穷尽搜索所有博弈树节点的 A 策略和有选择地搜索一部分节点的 B 策略。香农也因此成为计算机博弈的创始人。图灵写出了第一个国际象棋计算机博弈的程序。但由于当时的计算机成本高昂，尚未普及，而且运算能力也非常有限，图灵不能将自己的程序放在机器上运行。但是他想出了一个办法，就是把自己当成一个 CPU，并严格按照程序指令的顺序进行操作，平均每走一步就要花费半个小时左右。尽管当时的程序棋力和效率都非常低，但是科学家们这种契而不舍的精神鼓励着人们继续对计算机博弈进行研究。

随着计算机软硬件水平的高速发展，对国际象棋计算机博弈的研究也逐步取得进展，计算机博弈系统水平不断提高。

1957年，伯恩斯坦采用香农的B策略，设计出了一个完整的象棋程序，这个程序在IBM704上运行,从此第一台能进行人机对弈的计算机诞生了。这台机器的运算速度为每秒200步。

1973年，美国西北大学开发出来了CHESS4.0，成为未来程序的基础。1978年，CHESS4.7达到A级(相当于国际象棋一级)水平，1979年的更高版本CHESS 4.9夺得了全美国际象棋计算机大赛冠军，并达到了专家级水平(相当于国际象棋1-3段)。

1983年,肯.汤普森开发了一台专门下国际象棋的机器BELLE，可以搜索到八至九层，达到了精通级水平（相当于国际象棋4-6段）。

1993年,“深思”二代击败了丹麦国家队，并在与世界优秀女棋手小波尔加的对抗中获胜。

1997年，卡内基梅隆大学组成了“深蓝”小组研究开发出“更深的蓝”。这个计算机棋手拥有强大的并行处理能力，可以在每秒钟计算2亿步，并且还存储了百年来世界顶尖棋手的10亿套棋谱，最后“超级深蓝”以3.5比2.5击败了棋王卡斯帕罗夫。成为人机博弈的历史上最轰动的事件。

1.2 中国象棋计算机博弈研究现状

由于文化背景的差异，很少有西方学者对中国象棋计算机博弈问题展开研究，而国内计算机软硬件发展水平的落后，也成为制约中国学者进行研究的不利因素。因此,在国际象棋计算机博弈研究迅猛发展的时侯，对中国象棋计算机博弈的研究却十分滞后。无论是在博弈水平还是在研究规模上，都与国际象棋存在着差距。

对中国象棋计算机博弈的研究是从台湾开始的。当时可供参考的资料非常少,只能 借鉴国际象棋的成功经验。1981年台湾大学的张耀腾发表的硕士论文《人造智慧在电脑 象棋中的应用》，成为第一篇关于中国象棋计算机博弈的文章，虽然只用了残局来做实验,但是他介绍了评估函数的组成部分，并提出了当时的评估函数存在的问题,对以后的研究很有参考价值。

1982年台湾交通大学的廖嘉成在他的硕士论文《利用计算机下象棋之实验》中实现了一个完整的象棋程序，分为开局，中局，残局三部分，其中开局打谱，不超过二十步，中局展开搜索，残局记录杀着，已经具备相当的智能。而这种针对不同的对局阶段分别进行处理的方法也被现在的象棋软件普遍采用。

从1985年开始，台湾大学的许舜钦教授开始了全面的研究工作。在许舜钦1991年的论文《电脑对局的搜寻技巧》中总结了自1944年Von Neumann和Morgenstern提出的极大极小算法到当时为止最新发展的几乎所有算法，在他同年的另外一篇论文《电脑象棋的盲点解析》中从“审局函数偏差”和“搜寻深度不足”两大方面细致地论述了电脑象棋算法的7种误区。这些研究对计算机象棋的发展起了巨大的指导作用。许舜钦教授也因为自己的突出贡献而被称为“中国计算机象棋之父”。

随后对中国象棋计算机博弈的研究逐步展开。到目前为止出现了许多优秀的中国象棋软件，如台湾的许舜钦及其团队的“ELP”、赵明阳的“象棋奇兵″、中山大学涂志坚的“纵马奔流”、东北大学的“棋天大圣”、上海计算机博弈研究所黄晨的“象眼”等。其中东北大学人工智能与机器人研究所还专门聘请了“深蓝之父″许峰雄博士作为“棋天大圣”的顾问，获得过第十一、十二届电脑奥赛金牌和首届全国计算机博弈 锦标赛冠军。“象眼”是应用于“象棋巫师”上的搜索引擎，其创作者黄晨为了方便大家学习交流，公开了源码，并且发布了专门的网站作为计算机象棋知识和技术的交流平台。本文的程序就参考了一些相关技术。

作为中国的传统棋类游戏，中国象棋的空间复杂度比国际象棋高，规则也更为特殊，因而对它的研究也更具有挑战性。因此我们希望在目前中国象棋博弈系统研究成果的基础上，借鉴国际象棋的成功经验，总结出优秀的中国象棋搜索引擎有哪些共同点，尝试 新的研究方向，并且在一般配置的计算机上实现一个具有一定能力的中国象棋博弈系统，为今后中国象棋计算机博弈工作的深入开展提供帮助。

1.3 本文主要工作及难点

本文的主要工作有：

(1) 阐述并分析了基本模块以及搜索方法的基本原理。

(2) 提出并实现了一种基于评价函数的动态博弈树的新方法，测试、对比、分析结果。

(3) 设计并实现了一个较完整的中国象棋引擎（附录 1）。但由于作者水平有限，只能完成一定的残局挑战。

本文的主要难点有：

(1) 设计并实现引擎大体框架，保证框架无严重的漏洞，且有一定的效率。

(2) Alpha-Beta 算法的改进。如何在保证搜索不出错的情况下，加深搜索层次。

(3) 动态博弈树新方法的设计。如何对博弈树进行裁剪，使得能有效的提升博弈树搜索的效率且保证搜索不出错。

1.4 本文的结构安排

本文的主要研究内容是提升博弈树搜索算法的能力，并将其应用到中国象棋引擎上。本文共分为六章，各章节结构安排如下：

第一章，介绍课题的背景及意义、主要研究内容及难点。

第二章，阐述本文设计的引擎中三大基本子模块的基本原理。即局面数据结构、局面评价函数、着法排序的基本原理。

第三章，阐述以 Alpha-Beta 算法为中心的搜索方法的原理，分析增强博弈树搜索能力的方法，并展示测试结果。

第四章，阐述本文提出的一种新的提升博弈树搜索能力的算法的原理，展示测试结果，并与第三章的结果作对比。

第五章，总结全文的成果，并在此基础上指出下一步工作。

第二章 基本模块

2.1 引言

本文所设计的引擎的基本模块包括局面数据结构、局面评价函数、着法排序三个部分。基本模块为第四章所涉及的搜索方法提供了底层基础支持，搜索方法可以任意调用其中的外函数来局面消环、哈希判重、生成着法及递归搜索等等。基本模块是引擎的第一层，很多函数需要多次重复地被调用，因此，基本模块的效率至关重要，如果函数复杂度过高，会影响引擎的搜索能力。基本模块应当做到前后逻辑通顺、无严重的漏洞、高效率。

2.2 局面数据结构

局面数据结构部分主要包括棋盘表示、着法生成、哈希函数、置换表、回滚列表五个部分。其中棋盘表示是局面数据结构的核心，它负责描述棋盘局面；着法生成负责走棋，它能够高效的将一个局面转化为另一个局面。此外，哈希函数负责将局面单一的转化为计算机可识别的变量类型；置换表负责有效的存储及提供搜索过程中搜索过的局面。

本文的引擎采用面向对象的方法来表示棋盘，相关代码见附录 2。

2.2.1 局面表示

局面表示的作用是将棋盘数据转化为电脑语言，让计算机能够识别棋盘。因此，局面表示的关键在于转化方法，转化方法不仅要有逻辑性，还要有效率。

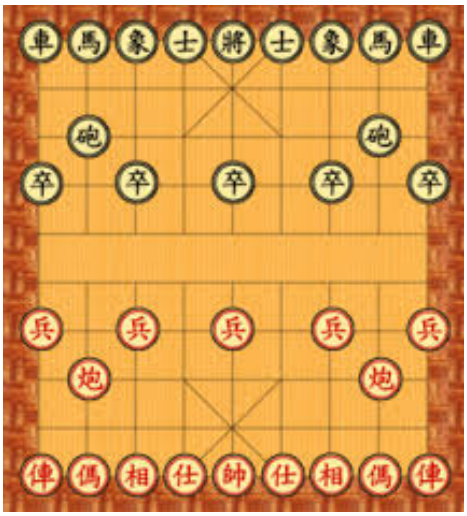

图 2-1 中国象棋开盘局面

设棋盘数组 square 来表示棋盘。中国象棋棋盘有 10 行 9 列，但本文不用 square[10][9]，而是用 square[256]来表示棋盘。将 10x9 的棋盘，转化为 16x16 的大棋盘，这样的设计有诸多好处。

图 2-1 为标准的中国象棋开盘局面，其转化后的 square 数组如下表：

表 2-1 square 数组

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	39	37	35	33	32	34	36	38	40	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	41	0	0	0	0	0	42	0	0	0	0	0
0	0	0	43	0	44	0	45	0	46	0	47	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	27	0	28	0	28	0	30	0	31	0	0	0	0
0	0	0	0	25	0	0	0	0	0	26	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	23	21	19	17	16	18	20	22	24	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

如表 2-1 所示，表第 1-3 行、14-16 行、1-3 列、13-16 列全为空行、空列。红方棋子数字从 16 到 31 表示，黑方棋子数字从 32 到 47 表示，这样的设计有如下好处：

(1) 不需要对棋子走出棋盘进行判断，只需判断 square[p]是否是 0 即可。将原本的“if i≥0 AND i＜10 AND j≥0 AND j＜9”四次判断转化为“if square[p]等于 0”一次判断。

(2) 方便用一个 16 进制整数存储每行每列的棋子位置状态。表 2-1 所示第 4 行的位置状态即为“0001111111111000”，第 4 列的位置状态即为“0001001001001000”。

(3) 设走子方为 p（p 为 0 表示红方，p 为 1 表示黑方），则 16+16*p+x（x 从 0 到 15）表示 p 方的第 x 个棋子

(4) 设棋子为 x，用 x&16 的结果即可判断棋子的颜色。若 x&16 为 16，则 x 为红色；若 x&16 为 0，则 x 为黑色。

(5) 设棋子为 x，用 x&15 的结果即可判断棋子的种类。

2.2.2 着法生成

着法生成的作用是通过合法着子将当前局面生成为另一个局面，搜索方法可以调用其中的外函数来生成吃子着法、非吃子着法、将军着法等。

设棋子位置数组 piece，piece 数组大小为 48，piece[x]表示第 x 个棋子所在的位置，piece 与 square 互为逆函数，即 square[piece[x]]=x。与表 2-1 对应的 piece 数组如下表：

表 2-3 piece 数组

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
piece	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
x	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
piece	199	198	200	197	201	196	202	195	203	164	170	147	149	151	153	155
x	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
piece	55	54	56	53	57	52	58	51	59	84	90	99	101	103	105	107

除车、炮外，其它棋子归为一类，生成着法的算法如下：

(1) 预生成每个棋子在每个位置能走到的位置集合 s

(2) 枚举起始位置 p1 的集合 s 的每个位置 p2

(3) 若 square[p2]不为 0 且 COLOR square[p1]等于 COLOR square[p2]，则返回(2)

(4) 若 square[p1]为象或者马，设相应的象脚或马脚位置为 p3，若 square[p3]不为 0，则返回(2)

(5) 通过从 p1 到 p2 的着法生成新局面，返回(2)

对于车、炮，一种简单的生成着法算法如下：

(1) 枚举 4 个方向，若枚举结束，则退出

(2) 设起始位置为 p1，往该方向的下一步位置为 p2

(3) 若 p2 超出棋盘，则回到(1)

(4) 若 square[p2]不为 0，则令 n++

(5) 若 square[p1]为车且 n 为 1，则通过从 p1 到 p2 的着法生成新局面

(6) 若 square[p1]为炮且 n 为 2，则通过从 p1 到 p2 的着法生成新局面

(7) 返回(1)

鉴于以上车炮的生成着法算法复杂度较高，本引擎采用行位数组 bitRow 以及列位数组 bitCol 来优化。bitRow[i]及 bitCol[j]分别表示 square 数组的第 i 行及第 j 列的位置状态（见 2.2.1(2)）。

2.2.3 哈希函数及置换表

哈希函数作用是将每个局面单一的无冲突的映射为计算机可识别的类型，方便快速判断重复局面。因此，哈希函数的关键在于无冲突，保证任意两个不同的局面可映射为两个不同的值。

本文采用经典的 zobrist 哈希法，原理是给不同类型棋子在不同位置赋值为随机的 64 位整形数，一个局面的 zobrist 值通过多个整形数相互异或而得到。zobrist 哈希法的好处诸多，尤其是在生成着子时，修改 zobrist 值很方便，只需进行三次异或运算即可。zobrsit 哈希法能在一定程度上保证哈希不冲突，然而，本文采用双重哈希法，再大大降低哈希冲突的可能性。生成两个 zobrist 值 zobrist1 及 zobrist2 来表示一个局面，代价是多花一倍的哈希所需时间。

置换表是博弈树搜索过程中的重要部分，作用是避免重复搜索，即搜索过的局面不搜索，直接返回上次搜索结果。置换表应当有一定的容量（本引擎的置换表大小为 256MB）。对可置换的局面的判断方法应当保证合理性，否则会将不该置换的局面置换，导致搜索出错。

本引擎的置换表含有以下元素：局面深度、局面哈希值、局面着法、局面分值、节点类型。节点类型分为 alpha 节点、beta 节点、pv 节点三种（见 3.2.1 节）。置换表的维护包括初始化、删除、清空、更新、查询五个部分。其中更新置换表在每个节点递归搜索结束时进行（见 3.2.1 节），即获得每个节点最优着法及最优分值时进行。只有当新局面深度比置换表存储局面深度大，或者相同但新局面分值比存储局面分值高时，才会更新置换表。

此节只以置换表的查询函数来阐述置换表的原理，查询函数算法如下：

函数参数：查询局面 s1，搜索窗口[alpha,beta]

函数作用：递归搜索前调用，避免重复搜索

(1) 设通过的 s1 的 zobrist 值得到应在置换表中的位置 p

(2) 设置换表 p 位置所存储的局面为 s2，s2 的局面分值为 v1

(3) 若 s1 和 s2 一致，则到(5)；否则到(9)

(4) 若 s1 的深度不小于 s2，则到(6)；否则到(9)

(5) 若 s2 的类型为 pv 类型，则返回 v1；否则到(7)

(6) 若 s2 的类型为 beta 类型且 v1≥beta，则返回 v1，否则到(8)

(7) 若 s2 的类型为 alpha 类型且 v1≤alpha，则返回 v1；否则到(9)

(8) 返回最低分值

2.3 局面评价函数

2.3.1 引言

评价函数是直接体现引擎下棋智慧的精华部分。评价函数不单单直接影响着引擎的棋力，还影响着搜索方法，准确的评价函数可使得博弈树搜索裁剪地更准确。在第五章中，本文提出了基于评价函数的动态博弈树的新方法，评价函数的好坏直接影响着动态博弈树的优劣。

在程序中，评价函数综合了大量跟具体棋类有关的知识。现从以下两个基本假设开始：

(1) 评价函数能把局面的性质量化成一个数字。例如，这个数字可以是对取胜的概率作出的估计。但是一般情况下不给这个数字以如此确定的含义，因此这仅仅是一个数子而已。

(2) 衡量的这个性质应该跟对手衡量的性质是一样的（如果引擎认为本方处于优势，那么反过来对方认为自己处于劣势）。真实情况并非如此，但是这个假设可以让我们的搜索算法正常工作，而且在实战中它跟真实情况非常接近。

评价可以是简单的或复杂的，这取决于程序中加了多少知识。评价越复杂，包含知识的代码就越多，程序就越慢。通常，程序的质量可以通过知识和速度的乘积来估计：

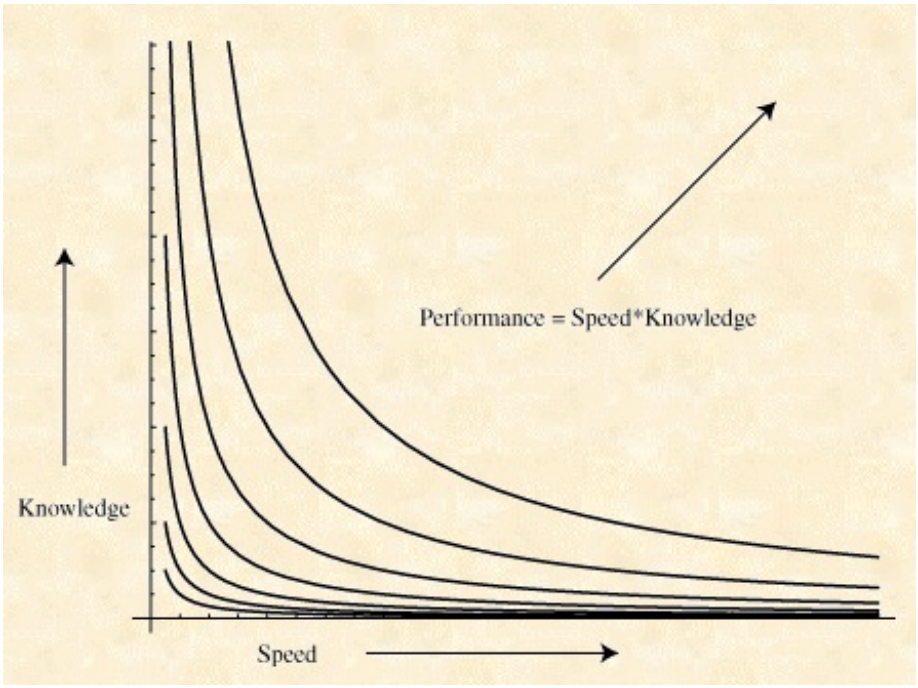

图 2-2 知识与速度的关系图

因此，如果有一个快速而笨拙的程序，通常可以加一些知识让它慢下来，使它工作得更好。但是同样是增加知识让程序慢下来，对一个比较聪明但很慢的程

序来说，可能会更糟；知识对棋力的增长作用会减少的。类似地，增加程序的速度，到一定程度后，速度对棋力的提高作用也会减少，最好在速度和知识上寻求平衡，达到图表中间的位置。平衡点也会随着你面对的对手而改变；对于击败其他电脑，速度的表现更好，而人类对手则善于寻找你的程序中对于知识的漏洞，从而轻松击败基于知识的程序。

本引擎的评价函数采用中国象棋对弈程序象眼（参考文献[2]）中的方法。主要评价是子力平衡评价，其次是车的灵活性以及马的阻碍。

2.3.2 子力平衡评价

子力平衡评价是通过评估双方每个棋子的权值，从而得到当前局面的分数。评估方法如图 2-3 所示，是通过求出本方权值以及对方权值，将二者相减从而得出当前局面分数。

$$V = V_{本方} - V_{对方}$$

$$V_{本方} = \sum V_{棋子i} + V_{调整}$$

$$V_{棋子i} = \frac{K_{中局} \times V_{中局} + K_{残局} \times V_{残局}}{K_{中局} + K_{残局}} \text{ (帅、马、车、炮、兵)}$$

$$V_{棋子i} = \frac{K_{进攻} \times V_{进攻} + K_{防守} \times V_{防守}}{K_{攻防最大}} \text{ (相、仕)}$$

$$V_{调整} = \frac{K_{象士防守} \times (K_{攻防最大} - K_{防守})}{K_{攻防最大}}$$

$$K_{中局} + K_{残局} = 66$$

$$K_{攻防最大} = 8$$

$$K_{象士防守} = 80$$

图 2-3 子力平衡公式组图

在本节，只阐述本方权值的求法，对方权值的求法类似。本方权值由两个部分组成，其一是所有本方棋子的权值之和，其二是权值调整值。本方棋子的权值要分两类来求，一类是帅、马、车、炮、兵，通过中局和残局的加权权值得到，

二类是相、仕，通过进攻和防守的加权权值得到。

由图 2-3 可知，要求得当前局面分数，只需求得中局比例值（$K_{中局}$）、进攻比例值（$K_{进攻}$）、防守比例值（$K_{防守}$）即可。而中局权值（$V_{中局}$）、残局权值（$V_{残局}$）、进攻权值（$v_{进攻}$）、防守权值（$v_{防守}$），均是根据棋子位置设定好的。

中局比例值通过计算所有本方棋子的带权和而得。帅不计，相、仕、兵计 1 分，车计 6 分，马、炮计 3 分。中国象棋一方有 2 个相、仕、车、马、炮，5 个兵，因此中局比例值与残局比例值之和为 66。计算出中局比例值后，残局比例值也随之而得。

进攻比例值通过计算本方过河棋子的带权和以及本方与对方马车炮数量差而得。本方过河棋子的权值分配是，马、车计 2 分，炮、兵计 1 分。计算双方马车炮数量差时，车的差值多算一倍。而防守比例值是以同样的方法但计算对方棋子的带权和以及对方与本方车马炮数量差而得。进攻比例值和防守比例值都分别需要与攻防最大值取最小值。

不同棋子中局权值、残局权值、进攻权值、防守权值，根据棋子的位置而设定。本节只以马的中局权值和残局权值来表述，如下表 2-4 及表 2-5：

表 2-4 马的中局权值分布

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	96	90	96	90	90	90	0	0	0	0
0	0	0	90	96	103	97	94	97	103	96	90	0	0	0	0
0	0	0	92	98	99	103	99	103	99	98	92	0	0	0	0
0	0	0	93	108	100	107	100	107	100	108	93	0	0	0	0
0	0	0	90	100	99	103	104	103	99	100	90	0	0	0	0
0	0	0	90	98	101	102	103	102	101	98	90	0	0	0	0
0	0	0	92	94	98	95	98	95	98	94	92	0	0	0	0
0	0	0	93	92	94	95	92	95	94	92	93	0	0	0	0
0	0	0	85	90	92	93	78	93	92	90	85	0	0	0	0
0	0	0	88	85	90	88	90	88	90	85	88	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

表 2-5 马的残局权值分布

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	92	94	96	96	96	96	96	94	92	0	0	0	0
0	0	0	94	96	98	98	98	98	98	96	94	0	0	0	0
0	0	0	96	98	100	100	100	100	100	98	96	0	0	0	0
0	0	0	96	98	100	100	100	100	100	98	96	0	0	0	0
0	0	0	96	98	100	100	100	100	100	98	96	0	0	0	0
0	0	0	94	96	98	98	98	98	98	96	94	0	0	0	0
0	0	0	94	96	98	98	98	98	98	96	94	0	0	0	0
0	0	0	92	94	96	96	96	96	96	94	92	0	0	0	0
0	0	0	90	92	94	92	92	92	94	92	90	0	0	0	0
0	0	0	88	90	92	90	90	90	92	90	88	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

2.3.3 车的灵活性及马的阻碍

在某些棋类中，棋盘可以分为一方控制的区域，另一方控制的区域，以及有争议的区域。例如在围棋中，这个思想被充分体现。而包括国际象棋在内的一些棋类也具有这种概念，某一方的区域包括一些格子，这些格子被那一方的棋子所攻击或保护，并且不被对方棋子所攻击或保护。在黑白棋中，如果一块相连的棋子占居一个角，那么这些棋子就不吃不掉了，成为该棋手的领地。空间的评价就是简单地把这些区域加起来，如果有说法表明某个格子比其他格子重要的话，那么就用稍复杂点的办法，增加区域重要性的因素。

除子力平衡之外，本引擎评价函数还包括了车的灵活性及马的阻碍，这二者正是考虑区域因素的体现。

车的灵活性指车往四个方向能够走到的空格数量，通过计算本方车的灵活性分值减去对方车的灵活性分值而得。

马的阻碍通过枚举马的八个可走到的位置，若位置上无棋子，且位置不受对方棋子保护，且位置不处于棋盘边界，则令 n++。若 n 为 0，则计负 10 分；若 n 为 1，则计负 5 分。马的阻碍也是通过计算本方马的阻碍分值减去对方马的阻碍分值而得。

2.4 着法排序

2.4.1 引言

Alpha-Beta 算法对着法排序非常敏感，所以采用启发式方法对其进行优化是提高算法效率的关键所在。本文引用参考文献[5]中的方法，并结合本引擎的实际情况，提出一种着法排序方案。在该方案中，将使用置换表启发、静态评价启发以及动态启发等技术。

2.4.2 置换表启发

在搜索中，经常会在不同的路径上遇到相同的棋局，这样的子树没有必要重复搜索，把最优着法、分值、深度和节点类型等信息保存在置换表中，再次遇到时直接运用即可。假设对某局面进行 d1 层搜索，窗口是[a, b]，而发现该局面在置换表中已存在，其评价值为 v，类型为 t，深度为 d2，当 d2≥d1 时，如果 t 为精确型，便可直接返回 v 而代替搜索；如果 t 为高出边界型且 v≥b，便可进行剪枝。否则进行重新搜索以保证所取得数据的准确，此时置换表中保存的最佳着法给了我们一定的启发，它为搜索提供一个较优着法，该着法应排在前面优先搜索，这使得总体着法顺序得到了一定程度的优化，置换表的一个主要作用是它对着法顺序的启发。值得一提的是，本引擎在使用置换表启发时，不考虑深度的影响，即只要查询局面在置换表中，则直接调用置换表中的着法。

2.4.3 静态评价启发

静态评价启发主要用来优化吃子着法的顺序。对吃子着法进行静态评价启发，就是要找出表面上占有较大优势的吃子着法。中国象棋的主要进攻手段就是吃子，首先通过检测吃子来寻找最优着法往往会产生较好的效果。如果我们用 V 表示攻击子的价值，用 W 表示被吃子的价值（各个棋子的价值如表 2-4 所示），那么某个吃子着法的价值 U 可表示为：

(1) $V-W$ （被吃子有保护）

(2) W（被吃子无保护）

表 2-4 棋子的交换价值

帅（将）	车	马	炮	仕（士）	相（象）	兵（卒）
5	4	3	3	1	1	2

当吃子着法的值 U＞0 时为表面上能得到获得很大利益的着法，这样的着法往

往是好的着法；U＝0 可能是等价值的换子，这样的着法也值得一试；而 U＜0 的着法往往是吃亏的。因此我们可将吃子着法依据 U 进行排序，并对 U≥0 的着法优先进行搜索。

2.4.4 动态启发

相对来说，对于中国象棋中的某一局面，U≥0 的吃子着法是很少的，大多数着法都是 U＜0 的吃子着法和非吃子着法。对于这些着法仅用之前所述的静态评价启发是不够的，还要进行动态启发。树搜索的过程，实际也是信息积累的过程。对这些信息进行挖掘，可以得到我们所需要的启发信息。

国际象棋的经验表明，历史启发和杀手启发都是很好的动态启发方法。历史启发的思想是：搜索树中某个节点上的好着法，对于其它节点可能也是好的。所谓好着法是指可以引发剪枝的着法，或者其兄弟节点中最好的着法。一经遇到这样的着法，算法就给其历史得分一个相应的增量，使其具有更高的优先搜索权，这个增量通常为$d\times d$（d 为当前节点需要搜索的深度）。具体地说，就是设立一个90×90的数组，红方和黑方的着法都记录在这个数组中，前一个指标代表起始格，后一个指标代表目标格；或者设立一个14×90的数组，红方着法和黑方着法分别记录，前一个指标代表兵种，后一个指标代表目标格。历史启发是从全局信息中获取优先准则的一种方法。对非吃子着法和表面上不能立刻得到实惠的吃子着法根据其历史得分进行排序，就能获得较佳的着法顺序。由于着法的历史得分随搜索而改变，对节点的排列也会随之动态改变。

杀手着法实际上是历史启发的特例。它是把同一层中，引发剪枝最多的着法成为杀手，当下次搜索时，搜索到同一层次，如果杀手是合法走步的话，就优先搜索杀手。根据国际象棋的经验，杀手产生剪枝的可能性很大，中国象棋也可以吸取这个经验。

2.4.5 结论及测试结果

针对中国象棋中的着法排列问题，本文将置换表着法、静态评价较优的着法和杀手着法排在前面，其余着法按历史得分依次降序排在后面，从而得到了一个较好的着法生成顺序。实验数据表明上述转化在同样的时间内使搜索时访问的平均节点数降低了一个数量级。

本引擎的着法排序流程示意图如下所示：

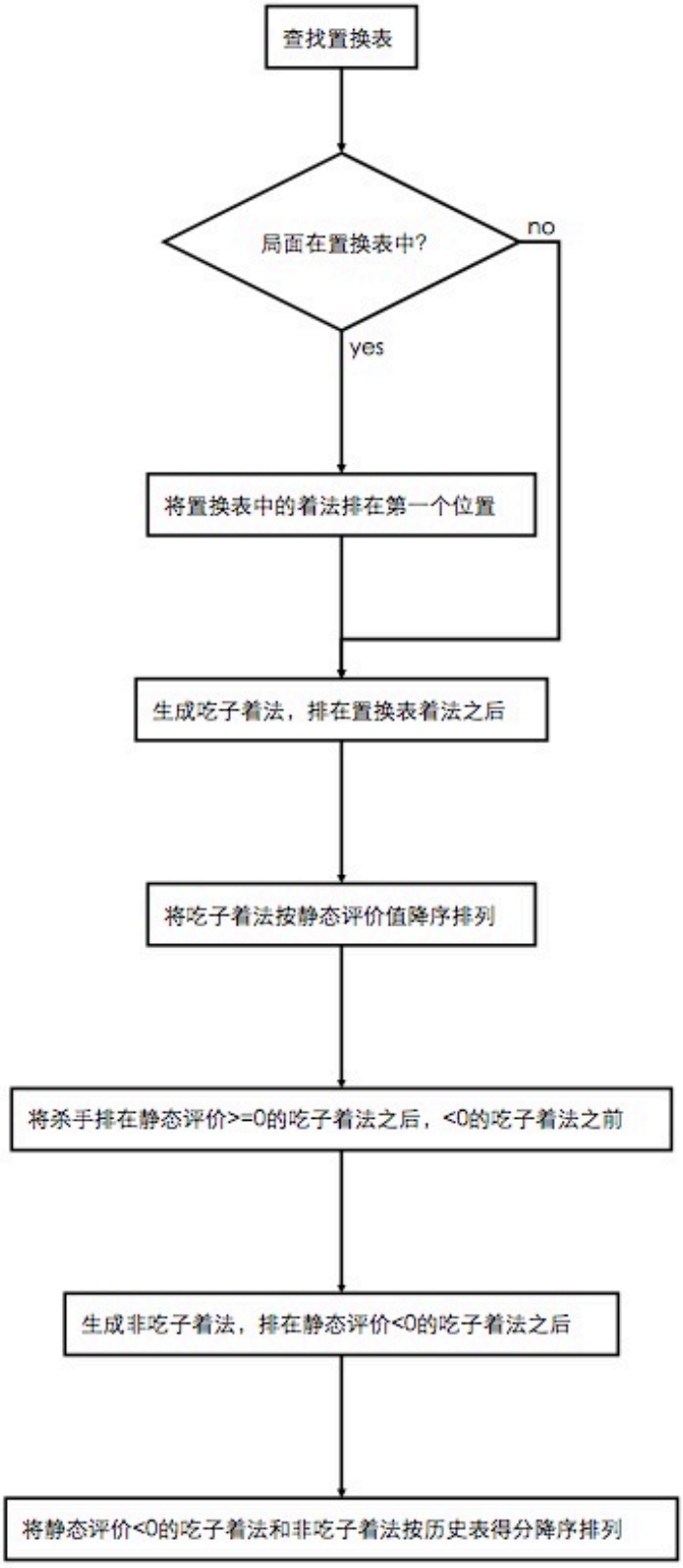

图 2-4 着法排序流程示意图

使用启发式方法对着法顺序进行优化可以大大减少搜索的节点数，实验结果如表 2-5 所示（测试数据选自参考文献[4]中有代表性的 10 个中局局面），其中优化前的着法顺序指的是以棋子为主键的排列顺序。从表 2-5 中可以看出随着搜索深度的增加，优化的着法顺序起的效用越来越大，这也验证了剪枝的效率与着法顺序是高度相关的。

表 2-5 不同着法顺序搜索访问的平均节点数对比

搜索深度	优化前的着法排序	优化后的着法排序	降低程度(%)
1	40	40	0
2	670	450	-33
3	5198	2766	-47
4	31408	10882	-65
5	263397	65516	-75
6	1422290	240260	-83
7	11400735	1257174	-88

值得指出的是，对杀手的位置进行了大量实验后，发现将杀手放在静态评价≥0 的吃子着法之后可获得更高的剪枝效率。表 2-6 对比了将杀手排在吃子着法之前与之后搜索时访问的平均节点数（测试数据选自参考文献[4]中有代表性的 10 个中局局面）。从表 2-6 中可以看出，杀手位置的调整对于搜索效率的提升是较为明显的。

表 2-6 杀手排在吃子着法前与之后搜索访问的平均节点数对比

搜索深度	杀手在吃子着法前	杀手在吃子着法后	降低程度(%)
1	40	40	0
2	502	450	-10
3	3400	2766	-18
4	14115	10882	-23
5	87275	65516	-25
6	332417	24260	-28
7	1798713	1257174	-30

2.5 本章小结

。

第三章 搜索方法

3.1 引言

。

3.2 Alpha-Beta 算法

。

3.3 主要变例搜索

。

3.4 迭代加深

。

3.5 静态搜索

。

3.6 结果测试

。

3.7 本章小结

。

第四章 残局挑战的测试结果

4.1 引言

。

4.2 建立动态博弈树

。

4.3 维护动态博弈树

。

4.4 测试结果并对比

。

4.5 本章小结

。

第五章 全文总结及展望

5.1 全文总结

。

5.2 后续工作展望

。

致谢

大学四年的生活即将结束，在学习过程中虽然遇到了很多问题，但是都在老师、家人、同学的帮助下顺利克服，在此对他们表示感谢。

首先，我要感谢我的指导老师——杨鹏老师。在完成毕业设计的过程中，杨老师对我进行了无私的指导与帮助，帮助我进行方法的改进与论文的完善，感谢您对我的帮助与鼓励。杨老师对学术对科研的严谨态度，使我一生收益。

其次，我要感谢上海贤趣信息技术有限公司，感谢你们的开源杰作象眼引擎以及象棋巫师网站，给我了很大的技术支持，在我屡次遇到困难时能有所依，有所靠。感谢你们对我的问题的回复，让我在完成毕设过程中扫清障碍，顺利前行。

感谢杨鹏教练以及校 ACM 集训队的同学。与你们一起在队内训练的两个暑假，使我终生难忘的经历，这使得我在编程与算法方面都有非常大的提高。

最后，感谢我的父母一直以来对我的支持与帮助，你们是我最坚强的后盾。

参考文献

[1] 上海贤趣信息技术有限公司.象棋百科全书网[EB/OL].http://www.xqbase.com/,Dec 15, 2015

[2] 上海贤趣信息技术有限公司.中国象棋对弈程序 ElephantEye(象眼)[CP/OL].https://github.com/xqbase/eleeye,March 16,2016

[3] 蒋鹏,雷贻祥,陈园圆.C/C++中国象棋程序入门与提高[M].北京:电子工业出版社,2009,1-333

[4] 黄少龙.象棋中局精妙战法[M].北京:金盾出版社,2004,94-105

[5] 岳金鹏,冯速.博弈树搜索算法在中国象棋中的应用[J].计算机系统应用,2009,(9):1-4

[6] 裴祥豪.基于剪枝策略的中国象棋搜索引擎研究[D].河北:河北大学,2009,6-8

[7] 本书编写组.象棋竞赛规则 201 试行[M].北京:人民体育出版社,2011,1-132

[8] 霍文会,王巍.象棋竞赛规则入门导读[M].北京:北京体育大学出版社,2006,1-181

附录

1. 引擎代码链接

https://github.com/peteryuanpan/lazyboy/tree/master/version0.2

2. 局面数据结构（部分）

```
struct PositionStruct {
    // 基本成员
    int player;   // 轮到哪方走，0 表示红方，1 表示黑方
    int square[256];   // 每个格子放的棋子，0 表示没有棋子
    int piece[48];    // 每个棋子放的位置，0 表示被吃
    int bitRow[16];   // 每行的位压
    int bitCol[16]; // 每列的位压
    std::pair<ULL, ULL> zobrist; // zobrist 值，双哈希
    int nDistance; // 搜索深度，初始值为 0
    bool check; // 将军态
    bool checked; // 被将军态
    bool chased; // 被捉态
    int vlRed; // 红方子力值
```

```
    int vlBlk; // 黑方子力值
};
```

外文资料原文

The History Heuristic and Alpha-Beta Search Enhancements in Practice

Jonathan Schaeffer

Abstract

Many enhancements to the alpha-beta algorithm have been proposed to help reduce the size of minimax trees. A recent enhancement, the history heuristic, is described that improves the order in which branches are considered at interior nodes. A comprehensive set of experiments is reported which tries all combinations of enhancements to determine which one yields the best performance. Previous work on assessing their performance has concentrated on the benefits of individual enhancements or a few combinations. However, each enhancement should not be taken in isolation; one would like to find the combination that provides the greatest reduction in tree size. Results indicate that the history heuristic and transposition tables significantly out-perform other alpha beta enhancements in application generated game trees. For trees up to depth 8, when taken together, they account for over 99% of the possible reductions in tree size, with the other enhancements yielding insignificant gains.

Introduction

Many modifications and enhancements to the alpha-beta (αβ) algorithm have been proposed to increase the efficiency of minimax game tree searching. Some of the more prominent ones in the literature include iterative deepening, transposition tables, refutation tables, minimal window search, aspiration search and the killer heuristic. Some of these search aids seem to be beneficial while others appear to have questionable merit. However, each enhancement should not be taken in isolation; one would like to find the combination of features that provides the greatest reduction in tree size. Several experiments assessing the relative merits of some of these features

have been reported in the literature, using both artificially constructed and application generated trees.

The size of the search tree built by a depth-first αβ search largely depends on the order in which branches are considered at interior nodes. The minimal αβ tree arises if the branch leading to the best minimax score is considered first at all interior nodes. Examining them in worst to best order results in the maximal tree. Since the difference between the two extremes is large, it is imperative to obtain a good ordering of branches at interior nodes. Typically, application dependent knowledge is applied to make a "best guess" decision on an order to consider them in.

In this paper, a recent enhancement to the αβ algorithm, the history heuristic, is described. The heuristic achieves its performance by improving the order in which branches are considered at interior nodes. In game trees, the same branch, or move, will occur many times at different nodes, or positions. A history is maintained of how successful each move is in leading to the highest minimax score at an interior node. This information is maintained for every different move, regardless of the originating position. At interior nodes of the tree, moves are examined in order of their prior history of success. In this manner, previous search information is accumulated and distributed throughout the tree.

A series of experiments is reported that assess the performance of the history heuristic and the prominent αβ search enhancements. The experiments consisted of trying all possible combinations of enhancements in a chess program to find out which provides the best results. The reductions in tree size achievable by each of these enhancements is quantified, providing evidence of their (in)significance. This is the first comprehensive test performed in this area; previous work has been limited to a few select combinations. Further, this work takes into account the effect on tree size of ordering branches at interior nodes, something not addressed by previous work.

The results show that the history heuristic combined with transposition tables provides more than 99% of the possible reductions in tree size; the others combining for an insignificant improvement. The history heuristic is a simple, mechanical way to order branches at interior nodes. It has none of the implementation, debugging, execution time and space overheads of the knowledge-based alternatives. Instead of using explicit knowledge, the heuristic uses the implicit knowledge of the "experience" it has gained from visiting other parts of the tree. This gives rise to the apparent paradox

that less knowledge is better in that an application dependent knowledge based ordering method can be approximated by the history heuristic.

外文资料译文

历史启发式和 Alpha-Beta 搜索增强实践

Jonathan Schaeffer

摘要

已经提出了对 alpha-beta 算法的许多改进以帮助减小极小树的大小。描述了最近的增强，历史启发式，改善了内部节点考虑分支的顺序。报告一组全面的实验，尝试所有增强功能的组合，以确定哪一个产生最佳性能。以前关于评估其绩效的工作集中在个人增强或少数组合的好处。但是，每个增强都不应该孤立; 人们希望找到提供最大减少树大小的组合。结果表明，历史启发式和转置表在应用程序生成的游戏树中显着地超出了其他 alpha beta 增强。对于深度为 8 的树，当它们合在一起时，它们占树的大小可能减少的 99％以上，其他增强效果不大。

简介

已经提出了对 alpha-beta（a-b）算法的许多修改和增强，以提高极小游戏树搜索的效率。一些文学中更突出的包括迭代深化，转置表，反驳表，最小窗口搜索，抽象搜索和杀手启发式。其中一些搜索工具似乎是有益的，而其他搜索工具似乎有疑问的优点。但是，每个增强都不应该孤立;人们希望找到能够最大限度减少树形大小的功能的组合。在文献中已经报道了一些评估这些特征的相对优点的实验，使用人工构建的和应用生成的树。

通过深度优先的 a-b 搜索构建的搜索树的大小很大程度上取决于在内部节点处考虑分支的顺序。如果在所有内部节点首先考虑导致最佳极小分数的分支，则出现最小 a-b 树。在最坏的情况下检查他们最好的顺序会导致最大的树。由于两个极端之间的差异很大，所以必须在内部节点获得良好的分支排序。通常，应用依赖的知识被应用于对订单进行“最佳猜测”决定以考虑它们。

在本文中，最近对 a-b 算法的增强，历史启发式进行了描述。启发式通过改进在内部节点处考虑分支的顺序来实现其性能。在游戏树中，相同的分支或移动

将在不同的节点或位置发生多次。维持每个举措在导致内部节点的最高极小分数方面取得成功的历史。不管起始位置如何，这个信息都是针对每个不同的动作进行维护的。在树的内部节点处，按照先前的成功历史进行检查。以这种方式，先前的搜索信息被累积并分布在整个树中。

报道了一系列实验，以评估历史启发式的表现和突出的α β检索增强。实验包括在国际象棋程序中尝试所有可能的增强组合，以找出哪些提供最好的结果。通过这些增强功能可以实现的树形尺寸的减少量被量化，提供了它们（重要性）的证据。这是在这一领域进行的第一次综合测试；以前的工作已经被限制在几种选择组合中。此外，这项工作考虑到在内部节点上订购分支的树形大小的影响，以前的工作没有解决。

结果表明，历史启发式结合转置表提供了超过99％的可能的树形大小的减少；其他组合改善不大。历史启发式是一种简单，机械的方式在内部节点上分配分支。它没有基于知识的替代方案的实现，调试，执行时间和空间开销。启发式使用隐含的知识，而不是使用显式知识，而是从访问树的其他部分获得的“经验”的隐含知识。这导致明显的悖论，较少的知识更好，因为基于应用的依赖知识的排序方法可以通过历史启发式近似。