河北大学

硕士学位论文

基于剪枝策略的中国象棋搜索引擎研究

姓名：裴祥豪

申请学位级别：硕士

专业：计算机软件与理论

指导教师：王熙照;翟俊海

20090501

摘 要

在人工智能（AI）领域，计算机博弈历来都是一个重要的研究方向。对中国象棋计算机博弈的研究始于上世纪八十年代，经过二十多年的努力，出现了大量优秀的博弈系统，在对弈能力方面，有些系统目前已经达到了专家级水平。然而与国际象棋所取得的成就相比仍有较大差距。

本文针对中国象棋博弈系统的搜索引擎进行研究，主要分析了各种基于剪枝策略的搜索算法应用于中国象棋博弈时表现出来的特点和性能，总结出了除算法之外其他影响系统棋力的因素以及改进的方法。B*算法很少被应用于中国象棋博弈系统之中，本文实现了基于最佳优先搜索的 B*算法，并设计了适合此算法的局面评估函数。在实验中详细分析了 B*算法的优缺点和实战能力，实验结果证明 B*算法应用于中国象棋博弈系统当中是可行的。

关键词： 计算机博弈　中国象棋　搜索引擎　剪枝　博弈树搜索　B*算法

Abstract

Computer game playing is a very important domain in Artificial Intelligence(AI). The research of Chinese Chess Computer Game started from the 1980s. After more than 20 years' development, many excellent game playing systems have emerged. Some of them have reached the human expert-level. However, comparing with the achievements of international chess systems, Chinese chess systems need to be improved.

In this thesis, the search engine of Chinese chess system is studied and the performance of search algorithms is analyzed. Some factors which can influence the systems' capability are studied and methods witch can improve the capability of the chess system are introduced. B* algorithm, witch is rarely used in Chinese chess system, is applied to our system. An evaluation function is constructed and combined with B*. The advantages, disadvantages and performance of B* algorithm are analyzed. Experimental results show that B* algorithm is feasible and effective in Chinese chess system.

Keywords: Computer Chess Game; Chinese Chess; Search Engine; Pruning; Game Tree Search; B* Algorithm

河北大学

学位论文独创性声明

本人郑重声明： 所呈交的学位论文，是本人在导师指导下进行的研究工作及取得的研究成果。尽我所知， 除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写的研究成果，也不包含为获得河北大学或其他教育机构的学位或证书所使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了致谢。

作者签名：裴祥豪 日期：2009 年 6 月 16 日

学位论文使用授权声明

本人完全了解河北大学有关保留、使用学位论文的规定，即：学校有权保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。学校可以公布论文的全部或部分内容，可以采用影印、缩印或其他复制手段保存论文。

本学位论文属于

1、保密 □ ，在_____年_____月_____日解密后适用本授权声明。

2、不保密 ☑ 。

（ 请在以上相应方格内打“ √ ” ）

保护知识产权声明

本人为申请河北大学学位所提交的题目为（基于统计的中医药检索引擎研究）的学位论文，是我个人在导师（张照，翟海）指导并与导师合作下取得的研究成果，研究工作及取得的研究成果是在河北大学所提供的研究经费及导师的研究经费资助下完成的。本人完全了解并严格遵守中华人民共和国为保护知识产权所制定的各项法律、行政法规以及河北大学的相关规定。

本人声明如下：本论文的成果归河北大学所有，未经征得指导教师和河北大学的书面同意和授权，本人保证不以任何形式公开和传播科研成果和科研工作内容。如果违反本声明，本人愿意承担相应法律责任。

声明人：裴祥豪 日期：2009 年 6 月 16 日

作者签名：裴祥豪 日期：2009 年 6 月 16 日

导师签名： 日期：2009 年 6 月 16 日

第 1 章 绪论

1.1 课题背景

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能的一种体现。

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，也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领域的一块试验田。

人们对机器博弈的研究主要集中在棋类博弈上[24]，包括国际象棋、跳棋、五子棋、西洋双陆棋（十五子棋）、围棋等等。其中对国际象棋计算机博弈的研究是规模最大、发展最为成熟的。

早在计算机发展的初级阶段，香农(1950)与图灵(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，但是科学家们这种锲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 IBM 704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，CHESS 4.7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）[6]。

80 年代中期，美国的卡内基梅隆大学开始研究世界级的国际象棋计算机程序——“深思”。

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[6]。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”[18]、东北大学的“棋天大圣”[20]、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[31]。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 所做的主要工作

棋类博弈问题实际上就是搜索博弈树的问题。中国象棋中局的每个局面都有大约四十种走法，其博弈树的规模大约是按四十为底的指数增长，随着搜索层数的加深，节点数目迅速增加，不但影响反应时间，还会超出计算机的承受能力。因此有必要对博弈树进行剪枝[4]。

利用算法剪枝是最常用的方法，目前已经有许多剪枝搜索算法[8]。除了利用算法本身的剪枝能力之外，还可以设计一些方法来辅助剪枝，比如优先展开利于剪枝的节点，优先寻找最佳路径等等。我们主要分析了剪枝算法加上辅助剪枝的方法组成的搜索引擎所具有的特点和性能。

B*算法是一种基于最佳优先搜索的算法，曾经成功的运用在十五子棋中，但却很少应用于中国象棋。我们实现了 B*算法在中国象棋当中的应用，并分析了算法的可行性。

针对 B*算法对估值的依赖性较强的特点，我们设计了专门配合 B*算法的评估函数，应用在系统当中，并就函数的合理性做了实验分析。

本文共分五章，第二章主要介绍了中国象棋计算机博弈的关键技术；第三章详细分析了剪枝策略在中国象棋搜索引擎中的应用；第四章介绍了 B*算法的应用；第五章是结束语。

第 2 章 中国象棋计算机博弈主要技术

一个完整的中国象棋搜索引擎开始工作，首先需要将棋盘和棋子在计算机中表示出来，然后需要生成博弈树，进行着法搜索，局面评价，并向界面返回最佳着法。如果在搜索引擎外部设置便于查询和操作的开局库和残局库，供搜索引擎调用，则可以提高系统的工作效率。我们依据这个顺序，分别从棋盘和棋子表示、着法生成、搜索算法、评估函数、开局库和残局库这几方面来介绍中国象棋计算机博弈的关键技术。

2.1 棋盘和棋子表示

中国象棋共有 32 枚棋子，其中红黑双方各 16 枚。这 16 枚棋子又分成 7 个兵种，不同的兵种具有不同的行棋规则。对每一种棋子进行编码，并区分出红黑双方，是最常用的方法（表 1）。

表 1. 棋子兵种编码表

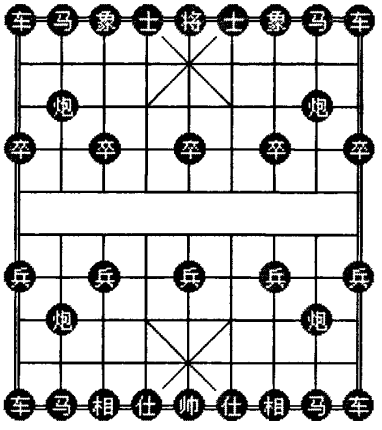
字母代号	K	B	E	K	R	C	P	—
棋子数目	1	2	2	2	2	2	5	—
红方棋子	帅	仕	相	马	车	砲	兵	无子
红方编码	1	2	3	4	5	6	7	0
黑方棋子	将	士	象	马	车	炮	卒	无子
黑方编码	8	9	10	11	12	13	14	0

表 1 中的编码方法只是对兵种做了区分。比如双方各有五个兵（卒），那么这五个棋子的编码就是一样的，而在对弈的过程中，不同的兵（卒）显然是不等价的，因此对同兵种的多个棋子赋予一个编码是不够的。为了更好的模拟棋子的挪动与拼杀，还需要对每一个棋子进行编码（表 2）。

表 2. 棋子编码表

编码	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
红方	帅	仕	仕	相	相	马	马	车	车	砲	砲	兵	兵	兵	兵	兵
编码	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
黑方	将	士	士	象	象	马	马	车	车	炮	炮	卒	卒	卒	卒	卒

中国象棋是在一个 10×9 的棋盘上进行搏杀的，因此最直观的表示就是用一个 10×9 的二维数组，或者用一个长度为 90 的一维数组来代表整个棋盘，数组中的每个单元代表棋盘上的一个交叉点，单元的内容代表该交叉点上的棋子编码，如图 1 所示：




```
Byte ChessBoard[90]={
12,11,10, 9, 8, 9,10,11,12,
 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0,13, 0, 0, 0, 0, 0,13, 0,
14, 0,14, 0,14, 0,14, 0,14,
 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0,
 7, 0, 7, 0, 7, 0, 7, 0, 7,
 0, 6, 0, 0, 0, 0, 0, 6, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0,
 5, 4, 3, 2, 1, 2, 3, 4, 5}
```

图 1. 中国象棋初始棋盘状态和棋盘编码

二维数组取坐标时需要做两次指针偏移，影响了效率，因此在实际应用中多用一维数组表示棋盘。这样的话，如果想得到某个棋子的横坐标，需要做一次求余运算，想得到纵坐标，需要做一次除法运算。由于计算机做除法的效率很低，而做位运算效率很高，借鉴国际象棋的棋盘表示方法，中国象棋也可以用一个长度为 256（16×16）的数组来表示，这样横坐标只用和十六进制的 F 求与，纵坐标右移四位即可，效率很高。

每走一步，对棋盘和棋子的操作只需要很少的时间（小于 1ms），考虑到剪枝策略的应用会节省大量时间，这个时间消耗可以忽略。因此我们为了突出算法的效率，采用了一维数组来表示棋盘，棋子也采用了简单的兵种编码。

2.2 着法生成

着法生成是指在当前局面下列举出所有符合规则的棋子走法。各种棋类由于行棋规则不同，走法生成的复杂程度也有很大差别。比如在五子棋中，某个局面下所有的空白交叉点就是该局面下的所有合理着法。因此只用扫描一遍棋盘，找到所有空白点就可以列举出所有合理着法。而中国象棋由于其规则的特殊性和复杂性，生成合理着法就需要大量的判断。比如要生成“马”的着法就需要判断是否蹩马腿，“象”的着法需要判断

是否堵象眼，“卒”过河之后不能后退等等。因此着法生成需要有一个合理性判断，分别对正常情况和应将情况下的着法进行判断。

博弈树是由着法和局面组成的，生成并执行了一个着法之后就会出现一个新的局面。因此着法生成实际上就是生成一棵以当前局面为根的博弈树。着法生成需要注意以下几个问题：

1）着法生成的效率

根据中国象棋规则定义可行区域，然后扫描棋盘，是着法生成最直观的方法。但是中国象棋由于规则复杂，着法生成时需要做大量的合理性判断，引起对棋盘相关区域的反复扫描，其时间消耗非常大，有时甚至能占到系统反应时间的一半左右。为此人们提出了许多优化方法，比如使用模板匹配法。

以“马”的着法生成为例，在正常情况下“马”有八种走法，如下图所示：

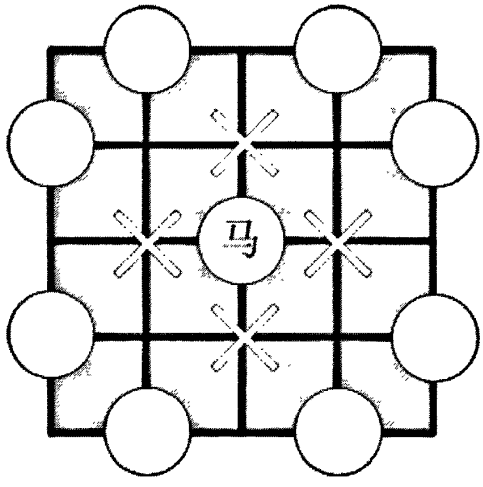


图 2. 马的着法匹配模板

在图 2 中，×代表马腿，○代表“马走日”的可能落址。在“马”的着法生成时，首先判断马腿处是否有子（马腿处的坐标可以在模板当中迅速计算出来），若马腿有子则相应的两个位置不能落子，马腿无子再去判断落址处是否己方棋子，若是己方棋子或是超出棋盘边界也无法落子，是对方棋子或是无子则生成相应的着法。下面这段伪代码是“马”的着法模板定义（棋盘数组为 1×90 的数组）。

```
int HorseLegTable[4] = {-1, 1, -9, 9};   //马腿处的坐标
int HorseMoveTable[8] = {-11, 11, -19, 19, -17, 17, -7, 7};   //正常落址
```

在“马”的着法生成时，以“马”的坐标与以上数组元素作和即可调出相应位置的坐标，通过判断该坐标处的棋子类型（无子为 0，红子 1-7，黑子 8-14）即可获知着法是否可行。

2）着法生成的空间复杂度

着法生成之后需要存储起来等待展开搜索。通常情况下系统会预先申请一块内存，避免频繁的动态申请造成不必要的时间消耗。那么需要预先申请多少内存才能满足系统的需要呢？

以中国象棋为例，对弈时每个局面下有大约 10 到 50 个合理着法，如果人为摆放的话可以有超过 100 个着法，但这种情况在正常的对弈过程中是不会出现的。一般来说 70 个着法就是最大值了，我们就取这个最大值。在基于极大极小搜索的博弈树中，由于采用了穷尽搜索的 A 策略，搜索的节点数最多，我们就在这种节点数最多的情况下考虑。如果程序可以搜索 6 层，每个着法要用 4B 的存储空间（一个整型变量），那么所需的内存总量是 70×6×4B=1680B，也就是 1.6K 左右的空间，由于我们考虑的都是最坏情况，可见这个空间需求量的绝对数值非常小。我们只用分配一块足够大的内存即可。

进一步分析空间复杂度，在穷尽搜索的一般情况下，假设一个局面之下的合理着法数是 N，搜索 M 层，每个着法需要 K 的存储空间，那么总的空间需求量就是 $N \times M \times K$。其中 M 是预先规定的常数，K 与数据结构有关，也是常数，因此总的空间需求量就是 $O(N)$，是一个线性的复杂度。

3）着法存储和展开顺序

一个局面下有很多合理着法，包括不同兵种的，不同位置的，吃子或非吃子的着法。这些着法的存储顺序在某些情况下会对剪枝搜索产生影响（具体分析见第三章）。着法存储下来之后，如果先对它进行排序，让吃子、兑子、将军等引起局面剧烈变化的着法优先被展开搜索，会提高一定的搜索效率。

2.3 搜索算法

搜索算法是搜索引擎的核心，在很大程度上决定了系统的搜索效率[25]。在一个局面之下直接判断着法的好坏并不精确，最好的方法是按照这个着法走下去，并进一步生成下一个局面的着法，继续深入展开搜索，比较走了若干步之后的情况。事实上人类棋手就是这样进行思索的，而考虑的步数越远，也就是博弈树的深度越深，棋手的水平就越

高。一个优秀的棋手往往能够看到四回合以后的结果，也就是大约七八层的深度，如果搜索算法也能达到甚至超过这个深度，那就可以和人类棋手进行高水平的对弈了。

2.3.1 博弈树搜索

中国象棋是一种零和博弈游戏，博弈双方交替出着。因此搜索的进程不仅取决于一方的意愿，同时还要参考对方的应对措施。在计算机博弈系统中博弈树可以看成是罗列了所有计算机和人类棋手可能着法和局面的一棵树。

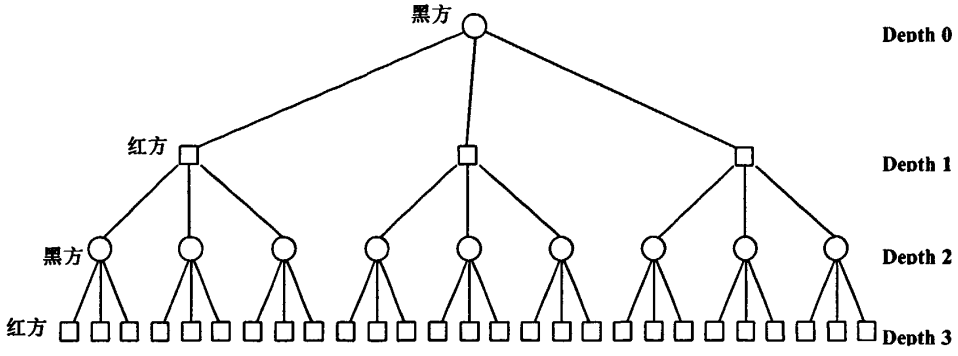


图 3. 博弈树示意图

图 3 表示一棵深度为 3 的完全博弈树。节点表示局面，两层节点之间的连线表示着法。其中根节点表示初始局面，叶子节点代表最终局面（胜、负或和），中间层的节点表示对弈过程中可能出现的合理局面。

博弈程序的任务就是搜索以当前局面为根节点的博弈树，找到从根节点到获胜或求和的叶节点之间的最佳路径，并返回这个最佳路径的第一步对应的着法。由于博弈树的搜索包含了对立的双方，一方只能做出一半的选择，对方的行动只能去猜测。从这个意义上看，博弈树是一棵与或树（AND/OR tree）。

如果能像图 3 一样建立起一棵完全博弈树，那么中国象棋计算机博弈的问题就可以解决[7]。但实际上这是不可能的，下面给出了各种棋类游戏的完全博弈树规模[19]。

表 3. 几种棋类的空间复杂度对比

棋类	状态空间复杂度	博弈树规模复杂度
国际象棋	50	123
中国象棋	**48**	**150**
日本将棋	71	226
围棋	160	400

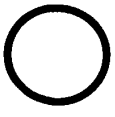
从表中可以看到，中国象棋的完全博弈树有10^{150}个节点，而据推算整个宇宙的原子数目也只有大约10^{80}个，也就是说即便我们可以生成这棵博弈树，并用一个原子存储一个节点的话，用上整个宇宙的物质都远远不够。

解决的办法就是，只把博弈树展开到一定深度，然后对这一深度下没有分出胜负的叶节点进行评估，以找到对自己尽量有利的着法。当然，博弈树展开的深度越深，离最终局面就越近，得到的结果就越精确，花费的时间也就越长。这就要求博弈系统在保证博弈水平的同时使反应时间尽可能短。

2.3.2 搜索算法的基础——极大极小算法

博弈树是由双方交替走棋产生的局面和着法共同构建的。假设轮红方走棋，在走了一步棋之后产生了许多后继局面，红方就希望在这些局面当中选择对自己最有利的一个。接下来轮黑方走棋，而黑方也希望在后继局面当中选择对自己最有利的一个，当然，这个局面对红方最不利。立足于红方考虑的话，如果给这些局面赋予一些评价值，红方选择评价值最大的局面，同时在下一步红方认为黑方会选择评价值最小的局面（这是基于双方有相等智力水平的假设）。将博弈树展开到一定深度，然后给叶节点赋予一些评价值，根据上面的假设进行搜索，就会得到根节点的值。这个过程就是 1950 年香农提出的极大极小算法的主要思路。

MAX 层



MIN 层

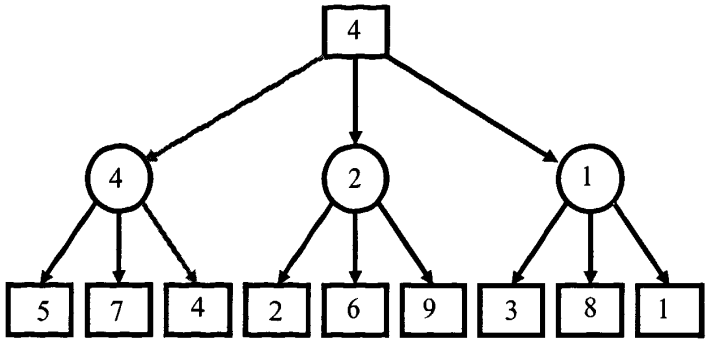


图 4. 极大极小算法示意图

在图 4 中，MAX 层取子节点中的最大者，MIN 层取子节点中的最小者。最大层和最小层交替作用，最终得到一条最佳路径（红线代表的路径），最佳路径的第一步就是在当前局面下要选择的最佳着法。下面是极大极小算法的伪代码

极大极小（*MAX – MIN*）算法过程：

```
int MAXMIN ( int depth )
  {
    if ( depth = = 0 ) return Evaluate ( );  //叶子节点返回估值
    if ( MAX 节点)//此句用于判断当前节点是 MAX 节点
      {
        for ( 每一步可能的着法 m )
            {
              执行着法 m;
              value = MAXMIN (depth - 1 );
              撤销着法 m;
              if  （ value > best ） best = value;
            }
      }
    if ( MIN 节点)//此句用于判断当前节点是 MIN 节点
      {
       for ( 每一步可能的着法 m )
            {
```

```
                执行着法 m;
                value = MAXMIN (depth - 1 );
                撤销着法 m;
                if （ value < best ） best = value;
              }
          }
      return best;
  }
```

算法中需要判断是极大节点还是极小节点。1975 年 Knuth 和 Moore 给出了负极大值（Negamax）的方法[1]，每一层都向上层返回负的极大值。负极大值实质上与极大极小算法是一致的，只是在形式上做了改变：

```
int Negamax ( int depth )
  {
    if ( depth = = 0 ) return Evaluate ( ); //叶子节点返回估值
        for ( 每一步可能的着法 m )
            {
              执行着法 m;
              value = - Negamax (depth - 1 ); //向上一层返回负的极大值
              撤销着法 m;
              if （ value > best ） best = value;
            }
  }
```

显然，形式上的小小改动令算法少做了两次判断，代码也更为简洁了。极大极小算法和负极大值法采用深度优先策略，对博弈树进行完全搜索，如果存在最佳着法，肯定能找到。极大极小算法是计算机博弈理论的基础。

2.3.3 搜索策略

除了极大极小算法之外还有许多搜索算法，这些算法将在后面的章节中出现。按照

对博弈树搜索策略的不同，这些算法可以分为三类：

1）穷尽搜索（Exhaustive search）：也就是香农提出的 A 策略，对博弈树进行完全搜索，极大极小算法就属于这一类。博弈树的规模庞大，在可接受的反应时间内穷尽搜索只能达到六层左右的深度，因此这类算法很少单独应用在博弈系统中。

2）选择性搜索（Selective search）：也就是香农提出的 B 策略，对博弈树剪枝，选择一部分着法展开搜索，比如 Alpha-Beta 算法，渴望搜索算法等。这一类算法是应用最为广泛的[27]。

3）启发式搜索（Heuristic search）：也叫目标导向搜索，制定一个目标作为搜索的指导方向，优先搜索与目标接近或可以更快达到目标的分枝进行搜索，如贪心搜索，分枝定界法等等。这类算法在中国象棋当中的应用比较少，本文研究的 B*算法就属于这一类[17]。

无论什么样的搜索策略，都是对一棵不完全的博弈树进行搜索。我们想对博弈树搜索的更深，就需要设计出更加合理的搜索机制，也可以采用其他的辅助手段让搜索算法能够发挥最大的效率。

2.4 评估函数

搜索算法依据局面评价值进行搜索，得到最佳着法。局面的评价值是搜索算法进行着法取舍的依据。在对弈的最后一个局面，由于胜负已经决出，可以给不同的结果赋予不同的值。比如获胜的情况赋值为 1，输了赋值为-1，下和赋值为 0。这些值都是精确值，如果我们能够得到这些值，那么返回的着法就一定是当前的最佳着法。但是由于我们不能将博弈树展开到分出胜负的情况，那就只能在博弈树展开到某一特定深度时，得到中间局面的分值，搜索算法依据这些中间局面的分值进行取舍。因此，我们需要一个评估函数，对中间的局面进行评价。

在搜索算法之外，评估函数是最重要的部分。因为局面评价的好坏，将直接影响到后面对局发展的趋势。评估过程对具体的棋类知识非常依赖，对于人类棋手而言，对局面评价的水平将直接决定他的对弈水平。人类大师凭借着丰富的经验和扎实的棋类知识储备，对局面的评价非常准确，才能走出不同于常人的妙招。

评估函数分为动态的和静态的，动态评估函数具有自学习的能力，一般是以神经网络的形式来表示，随着一盘盘的对弈，不断调整权重，使目标最优[13, 15]。动态评估函数

曾被成功的应用于西洋双陆棋的程序当中[3]。由于神经网络的结构选择和权重设置等问题无法从理论上解决，应用于中国象棋中的实际效果并不理想，其是否适合于中国象棋的局面评价仍在研究之中[22]。

目前最常用的就是静态评估函数[10]，静态评估函数的形式一般是一个多项式：

$$PositionValue = redValue - blackValue，\text{其中}$$

$$redValue(blackValue) = X_1 \times PieceValue + X_2 \times PiecePosition + X_3 \times Flexibility + X_4 \times Cooperation\ \&\ Threaten + Others$$

这个式子包括了子力价值、棋子位置、棋子灵活度、子力配合及受威胁程度等等因素。其他表示一些与中国象棋具体知识相关的因素。各项的系数由人为设定，一般取 1。这个式子并不是一个评估函数的标准形式，实际上关于评估函数的设计，由于和设计者的棋类知识密切相关，一直都是仁者见仁智者见智。但是有些部分已经取得了共识[16]，我们分别就这几类已经获得肯定的因素来讨论。

2.4.1 子力价值

子力价值是指每种棋子的固定值。棋盘上各兵种分工不同，重要性、灵活程度、走法和杀伤力的差异使得各种棋子的价值也不一样，比如一个车的价值比马大，炮的价值与马差不多等等。一般来说，在棋盘上，棋子多的占优势，棋子少的占劣势。由于中国象棋的规则是将死对方的帅（将）者获胜，因此给将帅分配一个远大于其他棋子的值是符合实际情况的。其他棋子根据经验，给出参考价值如表 4：

表 4. 中国象棋子力价值表

棋子类型	帅（将）	仕（士）	相（象）	车	马	炮	兵（卒）
子力价值	12000	240	240	1200	570	560	60

这些价值只能作为参考，因为不同的棋手对棋子价值定位的标准也是迥异的。有的人善于用马，觉得马的价值大于炮；而有的人善于用炮，又会认为用炮与对方换马对自己不利。有人提出用自学习的方法来确定子力价值，给每个棋子赋予相等的初始值，在与自己的对战中不断更新修改这些值，最终每个棋子收敛到一个比较固定的值[11]。这些标准都可以作为参考，但是计算机只能用一个标准来评判，考虑到炮在棋子较少时杀伤

力受影响，因此我们将炮的价值设得稍低于马。其他棋子的价值也是权衡之后给出的。

2.4.2 棋子位置值

相同的棋子在不同的位置其价值也是不一样的。最突出的就是兵（卒）在过河之后，因为行棋方式的变化，更靠近对方九宫，其价值会迅速增加。但是一旦兵（卒）到了底线，施加给对方的威胁就会立刻降低。下面是兵在棋盘上各可行位置的附加值：

0	3	6	9	12	9	6	3	0
18	36	56	80	120	80	56	36	18
14	26	42	60	80	60	42	26	14
10	20	30	34	40	34	30	20	10
6	12	18	18	20	18	18	12	6
2	0	8	0	8	0	8	0	2
0	0	-2	0	4	0	-2	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

图 5. 兵的位置附加值

其他棋子的情况也很类似，比如车占两肋需要加分，马卧槽需要加分，炮在中心线（当头炮）和底线（沉底炮）需要加分，等等。

2.4.3 棋子灵活度

如果棋子的活动范围很大，那么它的作用就会更容易发挥，因此棋子可移动方向和范围越大，相应的价值就越高。相反，如果棋子由于周围环境的限制导致攻击范围缩小或可移动方向减少，比如车被己方的棋子挡住，马被蹩腿，象眼被堵等等，那么它的作用就要大打折扣，相应的价值也要降低。

棋子灵活度对局势的影响相对于棋子自身价值来说并不大，也没有统一的计算标准，我们只能给出一个近似的计算公式：

$$FlexibilityValue = Sum(Num \times TypeValue)$$

其中 *Num* 表示有多少可走位置，*TypeValue* 表示某种棋子的固定价值，而 *Sum* 表示

整站优化 http://www.001baidu.com/

对所有棋子灵活度附加值求和。因此，*FlexibilityValue* 表示所有棋子灵活度的附加值之和。

对于棋子灵活度附加值最直观的解释就是，当一个棋子多了一个可行位置，它的附加价值就多一些。

2.4.4 子力配合及威胁

在中国象棋中，每一个的棋子都是作为本方所有棋子整体的一部分出现的，他们之间构成了某些关系。比如某个红兵落在了黑车的攻击路径上，同时红兵也在红马的合法落址上，此时黑车如果吃掉红兵，下一步就会被红马吃掉。这样一来，虽然黑方可以吃掉红兵，但由于一个车的价值远大于兵，用车换兵对黑方不利，此时黑车就不会吃掉红兵。而红兵由于被保护，虽然目前受到攻击，但是没有危险，因此可以不去考虑红兵受到的威胁。这属于一个棋子被保护的情况。

另外的情况是两个或多个本方棋子互相保护，比如连环马、担子炮、平头卒等等。由于两个棋子互相保护，此时可以考虑在他们受威胁的时候加上一些分值。

子力配合也要考虑轮谁走棋的问题。比如一个红马被黑车威胁，如果此时轮到红方走棋，黑马就可以躲开或是被其他棋子保护上，此时红方可以考虑少减分或不减分；如果轮到黑方走棋，则黑方就很有可能吃掉红马，此时红方就要减很多分。

2.4.5 其他因素

将帅所处的位置是一个很隐蔽的估值信息，如果对方在未被将的情况下移动将帅，那么很有可能是对方在接下来的几步中要利用将帅不能对脸的规则，对我方展开攻击，因此这个因素要考虑进来。此外，将帅附近的对方棋子过多，而本方在这一区域的子力价值总和并不大于对方，或者对方的棋子都能不受阻碍，直接对我方将帅进行攻击，也要考虑给予适当的减分。还有许多因素影响到评估函数对局面的评价，因为棋局是千变万化的，有些情况只在极小的概率下出现（比如在开局和中局等盘面子力较多的情况下人类棋手就很少走出不被将却挪动将帅的走法），给这些因素赋予多大的权重，有多大的概率会遇到这种情况，这些都是影响评估函数准确性的因素。

当然，即便是考虑了所有可能的因素，想设计出一个能精确地判定局面价值的评估

整站优化 http://www.001baidu.com/

函数，仍然不是一件容易的事情。设计者的棋类知识、实战经验都使得他们设计出的评估函数因人而异，各不相同。即便是同一个国际大师，在与不同的对手对战时对同一个局面的评价也是不同的。因此我们只是尽可能的让评估函数更能体现出局势的走向，尽可能的使评估函数更加人性化（比如用专家棋谱来训练它）[28,29]。

2.5 开局库

中国象棋的博弈一般分为开局、中局、残局三个阶段，所有的对局都必须有开局。

开局阶段一般是指在初始的 10-15 个回合。这个时侯双方各自调动兵力，占领棋盘的有利位置，摆好自己擅长的阵型。中国象棋出于其规则的特殊性，讲究迅速出动兵力，将各兵种棋子的特点发挥出来，中象开局可以形象的总结为“明车活马好炮位”[26]。

如果在一开始就调用搜索引擎进行搜索，那么我们至多可以得到搜索了十几层，相当于六七个回合之内的变化，很容易因为过分着眼于局部利益而影响了整体的局势。而且有些好的阵型和走法甚至不能搜索出来，比如第一步走三七路兵，搜索算法并不能找到这样的着法，但这是人们经过多年对战的经验总结出来的“仙人指路”局的第一步。中国象棋棋谱上记载的开局，都是经过了千锤百炼的经典着法，这些着法公认是对全局的发展大有裨益的。如果能将这些现成的开局着法存储起来加以利用，在开局只查找不搜索，会大大的提高计算机的反应速度和博弈水平，开局库就做了这样的事情。

开局库中存储了数以万计的经典局面和着法，如何保证在调用时快速匹配到当前局面呢？根据国际象棋的成功经验，最好的方法就是采用 *Zobrist* 哈希技术。通过一个求异或的转换公式，可以快速的找到开局库中的最佳着法。

2.6 残局库

对于残局，一般意义上的理解是子力较少的情况下就进入了残局，因此对中局和残局的区分就比较模糊。在具体的对战过程当中可能下到中局就决出胜负，从而没有了残局。与开局库的成功运用不同，人们对于残局库的研究并不系统。

最主要的原因就是残局的情况比开局复杂得多，在残局阶段可能出现的局面数是一个天文数字。肯.汤普森在上世纪八、九十年代已经成功的做出了国际象棋五子残局，只要棋盘上只剩下五个棋子，无论是哪些子，都可以在残局库中找到是胜、负还是和，而

整站优化 http://www.001baidu.com/

且给出将死的步数。但是五子的情况仍然太少，有的残局甚至多达十几个子，排列组合的迅速膨胀使做出更多棋子的残局成为不可能的事（目前甚至连完整的六子残局都做不出来）。因此将残局局面都存储下来加以匹配是不切合实际的。对多数残局的研究只能通过经验知识的运用，中国象棋的棋谱中记录了许多对残局的评判，并总结出了诸如单马八步擒单士，单车必胜单马，车兵必胜士象全等等规则，这些经验知识都可以应用于残局评判中，只要棋盘上剩下这些子，就可以直接判定胜负，停止搜索。

2.7 本章小结

这一章对中国象棋计算机博弈的关键技术作了介绍和简要的分析。目前对于搜索算法的研究已经非常成熟，也已经开始了对动态评估函数的研究。中国象棋的搜索引擎是由搜索算法、评估函数和着法生成器组成的，其实这几部分也是机器博弈的核心部分。一个优秀的搜索引擎都综合运用了多种技术[23]，但是共同的一点就是它们都应用了剪枝策略[21]。有了这一章的基础知识，我们接下来将介绍剪枝策略的应用。

整站优化 http://www.001baidu.com/

第 3 章 剪枝策略在中国象棋中的应用

由于博弈树规模过于庞大，我们需要进行剪枝。剪枝策略包括剪枝算法，启发式算法和其他一些辅助剪枝的手段，只要是有利于对博弈树进行剪枝的方法，就可以归入剪枝策略中，比如置换表、着法排序等。在这里剪枝有两个含义，最直观的含义就是剪去了冗余分枝，如 Alpha-Beta 剪枝等算法。另外，如果直接找到了最佳分枝，那么就可以停止搜索，实际上也等于剪掉了其余分枝。这一章我们从剪枝搜索算法、其他辅助手段等方面来介绍剪枝策略在中国象棋中的应用。

3.1 Alpha-Beta 算法

3.1.1 算法思路

在极大—极小算法搜索的过程中，存在着一定程度的数据冗余。如下图所示的博弈树片段。

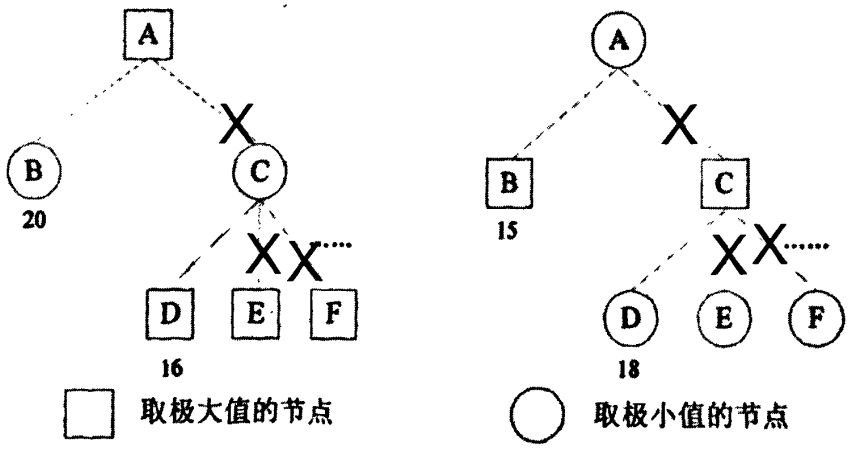


图 6. Alpha-Beta 剪枝示意图

整站优化 http://www.001baidu.com/

在图 6 左边，A 节点取子节点中的最大者，现在已经搜索过节点 B，得到其值为 20，于是知道 A 的值至少为 20。继续展开 C 进行搜索，C 取子节点中的最小者。得到 C 的子节点 D 值为 16，于是知道 C 的值至多为 16。由于 C 节点的值肯定不会大于 B 节点，也就是说 A 节点的值只与 B 有关而与 C 无关了。此时无论 C 的另外两个子节点 E 和 F 取值如何，都不会影响 A 的值，在搜索完 D 之后，可知 A 的最终取值为 20。上面的叙述可以用这个表达式表出：

$$Value(A) = Max(20, Min(16, E, F))$$

这个过程被称作 alpha 剪枝。

右边的情况与左图类似，不再累述，给出右图剪枝过程的表达式：

$$Value(A) = Min(15, Max(18, E, F))$$

这个过程被称作 beta 剪枝。这两个过程就是 Alpha-Beta 搜索算法的基本思想。

3.1.2 算法流程

将 alpha 剪枝和 beta 剪枝与极大极小算法的思想结合起来，就得到了 Alpha-Beta 搜索算法。下面是这个算法的类 C 伪代码，和极大极小算法一样，也应用了负极大值思想：

```
int AlphaBeta ( int depth ， int alpha，int beta )
  {
    if ( depth = = 0 ) return Evaluate ( )；//叶子节点返回估值
        for ( 每一步可能的着法 m )
            {
              执行着法 m;
               value = -AlphaBeta (depth－1，-beta ，-alpha，)；
               //应用负极大值思想
              撤销着法 m;
              if （ value > alpha ）
                  alpha = value ；//取最大值
                  if ( alpha >= beta ) break；//发生 beta 剪枝
            }
```


整站优化 http://www.001baidu.com/

```
        return alpha;
    }
```

3.1.3 实验结果与分析

我们比较一下没有剪枝的极大极小算法与 Alpha-Beta 算法的效果，下面的数据都是在指定深度下开局前十步的平均值：

表 5. 搜索节点数比较

搜索深度	1	2	3	4	5
MAX-MIN	42	1590	66014	2381804	92400000
Alpha-Beta	42	544	10387	146042	1812586

表 6. 平均每步反应时间比较（ms）

搜索深度	1	2	3	4	5
MAX-MIN	1	80	4020	150210	6511451
Alpha-Beta	5	22	665	6371	120525

通过比较我们发现，极大极小算法由于采用穷尽搜索的策略，在搜索到第五层时其反应时间已经长达一个多小时，搜索节点数也接近一亿个，算法的效率很低。而 Alpha-Beta 算法则明显好于极大极小算法，尤其是在三层以后，节省的时间非常可观。

对于每一个被剪掉的节点来说，都意味着不仅节点本身，而且以这个节点为根的子树也被忽略掉了。这使得 Alpha-Beta 算法搜索的节点数远远小于极大极小算法。而 Knuth 和 Moore 所做的研究证明了 Alpha-Beta 算法虽然剪掉了一些分枝，忽略了许多节点，但是并没有剪掉最佳分枝，在搜索结果上与极大极小算法是一致的[1]。极大极小算法搜索的节点数 M 是这么计算的：

$$M = W^d$$

其中 W 代表每个节点的平均分枝数，d 代表搜索深度。而在最好情况下，Alpha-Beta 算法搜索的节点数 N 是这么计算的：

$$N = W^{(d+1)*2} + W^{d/2} + 1$$

也就是说，大约是极大极小算法搜索节点数平方根的二倍。

整站优化 http://www.001baidu.com/

3.2 渴望搜索算法

3.2.1 算法思路

从 Alpha-Beta 算法的流程可以看到，窗口 $(alpha,beta)$ 一直在收敛，发生剪枝的条件是下界 alpha 超越上界 beta。而算法一开始所用的窗口是 $(-\infty,+\infty)$，这就使收敛的速度受到影响，从上一节的实验结果可以看到 Alpha-Beta 算法搜索到五层深度时反应时间就达到了两分多种，这样的效率在实际对战中不太能够让人接受。我们可以这样设想：如果在一开始就指定一个小一些的窗口来代替 $(-\infty,+\infty)$，是不是可以让算法更快的发生剪枝呢？

渴望搜索（也叫期望搜索）算法就是基于这样的思想，从一开始就在一个小窗口 (A,B) 之内进行搜索，试图加快剪枝进程。A 和 B 的值是这样确定的：首先调用评估函数对根节点估值，设此值为 V，取 $(V-value,\ V+value)$ 作为初始窗口。$value$ 在国际象棋中通常取半个兵的价值，考虑到中国象棋中的兵作用与之相似，我们也取半个兵的价值。显然，当 $value$ 的值很小的时候搜索窗口的宽度就会非常小，可能会引发更快的剪枝。

在这里我们用了“可能”一词。因为根节点的最终返回值不一定正好就落在这个小窗口中。这个真实值，把它记做 $best$，有三种可能性：

1）$best \in (A,B)$。此时 $best$ 就是根节点的真实值，会很快的发生剪枝，这也是我们希望得到的结果。

2）$best \le A$。这种情况下，根节点的值肯定小于等于 $V-value$，但具体是多少，我们不知道。为得到根节点的真实值，我们需要用窗口 $(-\infty,best)$ 再次搜索。

3）$best \ge B$。这种情况下，根节点的值肯定大于等于 $V+value$，但具体是多少，我们不知道。为得到根节点的真实值，我们需要用窗口 $(best,+\infty)$ 再次搜索。

3.2.2 算法流程

我们将上面的三种情况分别加以处理，结合基本的 Alpha-Beta 算法，将渴望搜索

整站优化 http://www.001baidu.com/

用类 C 伪代码描述出来：

```
int alpha = V-value;
int beta = V+value;
value = AlphaBeta （depth，alpha，beta）;
        //此处的 AlphaBeta 即指调用 alpha-beta 搜索算法
if （value >= beta） value = AlphaBeta （depth，value，+∞）;
                    //高出窗口上边界(Fail high)
if （value <= alpha）value = AlphaBeta （depth，-∞，value）;
                    //低出窗口下边界(Fail low)
```

3.2.3 实验结果与分析

我们比较一下渴望搜索与 Alpha-Beta 算法的效果：

表 7. 搜索节点数比较

搜索深度	1	2	3	4	5
Alpha-Beta	42	544	10387	146042	1812586
渴望搜索	72	695	1088	41305	937988

表 8. 反应时间比较（ms）

搜索深度	1	2	3	4	5
Alpha-Beta	5	22	665	6371	120525
渴望搜索	1	40	60	2480	57025

渴望搜索明显比单纯的 Alpha-Beta 算法结果要好一点，这表明我们的猜测值在多数情况下总是落入了期望的区间的。在深度超过三层以后，Alpha-Beta 算法与渴望搜索的节点数与反应时间的比值降低，说明随着深度的增加，棋局变化复杂起来，实际估值落入猜测区间的可能性也降低了。但仍然能够节省大约一半的时间。

整站优化 http://www.001baidu.com/

3.3 最小窗口搜索算法

3.3.1 算法思想

就像当时设想渴望搜索一样，如果我们设想的更大胆一些，用一个更小的窗口进行搜索，是不是会产生更好的剪枝效果呢？

最小窗口搜索就是基于这样的思想。此算法假设任意局面下的第一个着法就是要寻找的最佳着法，其后继着法则次之。在一个中间节点，其第一个分枝以一个完整的窗口$(-\infty,+\infty)$搜索并产生一个位于该窗口内的值 V，则此节点的后继分枝使用一个最小的窗口$(V,V+1)$搜索之。最小窗口搜索的意图在于使用最小的窗口建立最小的博弈树，以此达到高效的搜索。当使用最小窗口$(V,V+1)$对后继分枝进行扩展时，情况等同于期望搜索了，当然，此时我们仍然要考虑高出边界和低出边界的情况。

3.3.2 算法流程

下面给出最小窗口搜索算法的类 C 伪代码：

```
int MinWindow（int depth，int alpha，int beta）
    {
      if （当前被探测节点为叶节点）return Evaluation（）；
      if （当前被探测节点为 MAX 节点）
          {
             生成此 MAX 节点的第一个着法 m；
             best = MinWindow（depth-1，alpha，beta）；
             撤销着法 m；
             for（从第二个着法起，所有剩余的着法 n）
                 {
                   if（best >= beta ）break；//此处为 beta 剪枝
                   else
```


整站优化 http://www.001baidu.com/

```
                    {
                      if（best > alpha）alpha = best;
                      执行着法 n;
                      value = MinWindow（depth-1，alpha，alpha+1）；
                      if（alpha < value && value < beta）
                    //表明第一个着法为最佳着法的假设不成立，需要对此着法
                   //重新进行扩展，以求找到最佳着法
                        value = MinWindow（depth-1，value，beta）；
                      if（value > best）best = value;
                      撤销着法 n;
                    }
                  return best;
              }
        }
if （当前被探测节点为 MIN 节点）
   {
      生成此 MIN 节点的第一个着法 m;
      best = MinWindow（depth-1，alpha，beta）；
      撤销着法 m;
      for（从第二个着法起，所有剩余的着法 n）
         {
           if（best <= alpha ）break；//此处为 alpha 剪枝
           else
                {
                  if（best < beta）beta = best;
                  执行着法 n;
                  value = MinWindow（depth-1，beta-1，beta）；
                  if（alpha < value && value < beta）
                //表明第一个着法为最佳着法的假设不成立，需要对此着法 n
                //重新进行扩展，以求找到最佳着法
```


整站优化 http://www.001baidu.com/

```
                    value = MinWindow（depth-1，alpha，value）；
                    if（value < best）best = value;
                    撤销着法 n;
                }
            return best;
        }
    }
}
```

3.3.3 实验结果与分析

比较一下渴望搜索与最小窗口搜索的效果：

表 9. 搜索节点数比较

搜索深度	1	2	3	4	5
渴望搜索	72	695	1088	41305	937988
极小窗口	47	562	8455	74898	785402

表 10. 反应时间比较（ms）

搜索深度	1	2	3	4	5
渴望搜索	1	40	60	2480	57025
极小窗口	10	40	505	4586	47795

通过比较发现，最小窗口有一定的风险。在三层和四层深度时，甚至不如渴望搜索。这是因为初始窗口太小，算法经常会出现返回值越界，一旦越界，就要重新进行搜索，重新搜索等同于渴望搜索，之前的搜索已经花费了时间，因此节点数和反应时间就比渴望搜索要多一些。但是当搜索深度继续加深时，随着局面的相似程度越来越高，使用最小窗口的优势就体现出来了，在第五层时明显的优于渴望搜索。这表明我们对局面进行大胆的猜测是有道理的。

整站优化 http://www.001baidu.com/

3.4 着法顺序

我们已经知道了 Alpha-Beta 算法在最好情况下的效率非常高，它评估的节点数是极大极小算法搜索节点数平方根的二倍。那么在其他情况下呢？我们来看看 Alpha-Beta 算法的另外一种情况：

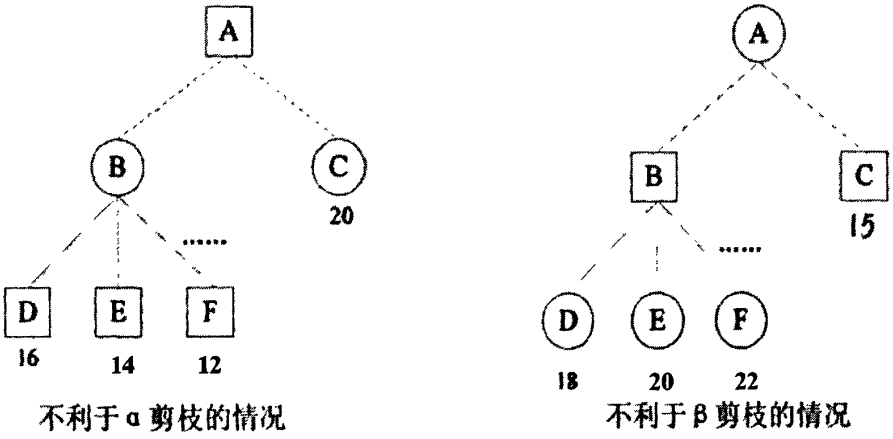


图 7. 不利于 Alpha-Beta 剪枝的情况

在图 7 左边，A 取 B、C 节点中的最大者，在搜索了以 B 为根的整棵子树之后，得到 B 的值为 12。然后再去搜索 C，发现 C 的值 20 比 B 要大，此时选择 C 代表的着法作为最佳着法。但此时已经搜完了整棵博弈树，也就是说在最坏情况下 Alpha-Beta 算法不发生剪枝，与极大极小算法一样。而如果先搜到 C 节点，那么只用在 B 的第一个子节点上进行一次搜索，就可以剪掉 B 节点，并返回 C 为最佳着法了。右边的情况可以类似推得。

由此可见，先展开哪个节点，对算法的效率有很大的影响[12]。因此，如果能在算法执行搜索之前就对所有着法排序，那么就能更充分的发挥 Alpha-Beta 算法的作用了。着法排序机制有很多，下面介绍两种常用的对着法进行排序的方法，迭代加深和历史表。

整站优化 http://www.001baidu.com/

3.4.1 迭代加深

在限定深度的搜索算法执行过程中，很可能会因为搜索深度不够而做出错误的判断。比如限定算法深度是七层，在第七层处计算机棋手执黑看到一个红马处于黑车的攻击路线上，于是返回吃马着法，但如果红马处于红车的保护之下，那么在第八层红车就会将黑车吃掉。由于算法只搜索到七层就返回了，计算机棋手并不知道在第八层会丢掉自己的车，从而吃一个马丢掉一个车，导致局面被动。这类现象被称为“水平效应”。

为了克服水平效应带来的不利影响，有许多搜索策略被提出来，比如静止搜索、选择性延伸等等。这些方法与剪枝无关，只是为了提高棋力而选择性的多搜索了几层，我们关注的是有利于剪枝的迭代加深法。

严格说来，迭代加深并不是一个独立的算法，而是一种控制搜索时间的机制。在限时走步、限定搜索深度的情况下，我们规定计算机棋手至多可以每分钟走一步，而实际上计算机棋手可能在 30 秒钟时就已经完成固定深度的搜索了，并找到了一个可行着法，但不一定就是当前局面下的最佳着法。如果我们把剩下的 30 秒钟也用上，让计算机棋手逐步加深搜索，就会在有限的时间里走出尽可能好的着法。

下面给出了迭代加深过程的类 C 伪代码：

```
for（depth = 1； depth <= MAXdepth； depth + +）
    {
      value = AlphaBeta（depth， alpha， beta）；
      if（TimeOut（））break；
      //TimeOut（）函数用于检测搜索是否超时
    }
```

在此引入迭代加深还有一个更重要的目的，就是利用搜索过的最佳着法来指导搜索的方向。当我们完成了 d 层的搜索之后，如果时间有剩余，还要进行第 d+1 层的搜索。此时在 d 层找到的那个最佳着法，就可以作为 d+1 层搜索时优先展开的分枝。因为相邻的局面具有一定程度的相似性，上一层的最佳着法也有可能是这一层的最佳或次佳着法，根据这些已获得的知识指导着法按顺序展开，就会提高搜索的效率。

下面比较了利用迭代加深的 Alpha-Beta 算法和基本的 Alpha-Beta 算法的效果

整站优化 http://www.001baidu.com/

表 11. 搜索节点数比较

搜索深度	1	2	3	4	5
Alpha-Beta	42	544	10387	146042	1812586
Alpha-Beta+迭代加深	42	465	10350	144802	1810152

表 12. 反应时间比较（ms）

搜索深度	1	2	3	4	5
Alpha-Beta	5	22	665	6371	120525
Alpha-Beta+迭代加深	5	20	665	6373	120520

通过比较我们发现，两种方法的效率不相上下，应用了迭代加深的算法要稍微好一点，仅就搜索节点数而言，也没有起到很明显的剪枝作用。于是我们让这两个版本的程序对战，限定五秒钟走一步，在十局对局中，应用了迭代加深的算法以七胜三和的成绩击败了三层搜索深度的 Alpha-Beta 算法。据此分析，迭代加深算法确实产生了良好的剪枝效果，但由于加深了搜索层数，等于又额外搜索了一些节点，因此总体上节点数目与 Alpha-Beta 算法相差无几，也正是因为搜索层数较深，使应用了迭代加深的程序搜到的着法更好，棋力更强。

3.4.2 历史表

从迭代加深的思想出发，如果我们将以前搜索到的最佳着法都存储下来，并认为在接下来的局面中一旦遇到这些着法是可行着法，也假定它们是新局面下的好的着法，优先展开进行搜索。

好的着法究竟该打多少分？对于这个问题有两种考虑。一是认为搜索了较深层数得到的着法要比一个浅层搜索得到的着法好，二是认为博弈树被搜索的越深，树中任意两个局面的相似度就越小，靠近根节点的好的着法被重复搜索的机会就越大。基于以上两点，可以使用 2^{depth} 给被再次搜索到的好的着法加分。

将着法映射到历史表中，需要用一个二维的数组，第一维表示初始位置，第二维表示目标位置，历史表的大小一般取 256×256。

我们将历史表与 Alpha-Beta 算法结合，并与基本的 Alpha-Beta 算法作比较：

整站优化 http://www.001baidu.com/

表 13. 搜索节点数比较

搜索深度	1	2	3	4	5
Alpha-Beta	42	544	10387	146042	1812586
Alpha-Beta+历史表	41	255	2645	19768	179845

表 14. 反应时间比较（ms）

搜索深度	1	2	3	4	5
Alpha-Beta	5	22	665	6371	120525
Alpha-Beta+历史表	4	21	163	1056	11897

通过对比发现，应用了历史表之后，Alpha-Beta 算法的效率确实提高了，而且层数越深，提高的程度就越明显，在第五层已经缩短了十倍左右的反应时间，节点数也急剧减少到原来的十分之一左右。这再一次证明了 Alpha-Beta 算法对着法展开顺序的敏感性。

3.5 本章小结

在这一章中，我们对主要的剪枝算法和辅助剪枝方法进行了分析和对比。剪枝之后的博弈树规模变小，使博弈系统可以搜索更多更深层次的着法，在提高了程序效率的同时提高了系统的棋力。辅助方法的应用需要占用一定的存储空间，但它可以使搜索算法更有效的工作，相对于棋力上的提高，这些存储空间的耗费是值得的。下一章将介绍我们工作的主要内容，能够直接找到最佳分枝的算法——B*算法在中国象棋博弈系统中的应用。

整站优化 http://www.001baidu.com/

第 4 章 B*算法的应用

B*算法是 Berliner 在 1979 年提出来的一个算法。1979 年 7 月 15 日，由 Berliner 编写的十五子棋（也就是西洋双路棋，Backgammon）程序在蒙特卡洛的一次比赛中以 7：1 的悬殊比分击败了当时的世界冠军 Luigi Villa，从而在相当智力水平的棋类比赛中，计算机棋手第一次击败了人类棋手。

B*算法没有采用极大极小思想进行搜索，而是基于分枝限界的思想选择最优节点。与分枝限界法不同的是，B*算法返回修正上下界的值，每一个节点的上下界都动态可变。目前国内对 B*算法的研究比较少，只能在陆汝钤老师的《人工智能》上看到比较全面详细的描述[14]。

4.1 算法思想

在极大极小博弈树中，我们每次都预测自己和对方的反应，然后选择符合极大极小原则的分枝作为最佳着法。对于自己的反应很容易预测，为了赢棋，我们选择评估值最大，也就是对自己最有利的那一枝。而对方的想法我们无法得知，我们只能想当然的假设，对方有着和我们一样的智力水平，因此对方也会选择对他自己最有利的那一枝，我们把这一分枝叫做 B。可事实上这种假设不一定符合实际情况：对方如果没有我们聪明，那么他就不会选择 B，对方如果比我们水平高，那么他有可能设下陷阱，也不会走 B 这一枝。我们之前的搜索是在以 B 为根的子树上进行的，如果对方选择其他分枝，比如选择了 A，那么接下来轮我们走棋的时候我们需要重新搜索子树 A，之前在 B 子树下搜索过的节点信息就无法利用，此时无论是迭代加深还是历史表都起不到很大作用了。

Berliner 针对这一问题进行如下分析：

1）Berliner 认为，普通的博弈树搜索一定要找到从当前状态到目标状态（也就是限定深度下的叶节点）的全路径后才能结束。而实际上只用返回这个全路径的第一步。因此，如果只做方向性的搜索，确定如何朝最佳目标的方向迈出第一步就行了。在博弈树搜索中存在着“对方将如何反应”的不确定因素，也不可能将博弈树搜索到底。

2）对搜索方向进行预测时，需要定量的估计，但不用给出具体的估值。因此需要

整站优化 http://www.001baidu.com/

设计一个估值区间，能包括某一枝估值的可能范围。考虑到对方可能很笨，也可能很聪明，因此我们分别使用两个估值函数——一个乐观估计和一个悲观估计来确定这个区间是比较合适的[30]。

3）利用启发性因素比盲目的最佳优先策略要灵活一些，因为最佳优先总是选择具有最佳估计值的那一枝，而从不考虑其余分枝的情况。实际上我们有两种方法来确定一个分枝是最佳的：一种是把这个分枝一搜到底（直接证明），另一种是把其余分枝展开，并证明他们不是最佳的（间接证明）。

4.2 博弈树搜索过程

B*算法进行博弈树搜索时，对每个节点都进行乐观估计和悲观估计，体现对当前局面的预测趋向。两个估计值都动态可变，而且所有节点上的估计值出自同一方的立场，但估计的对象按层次交错更替。比如这一层是对本方棋局的估计，下一层是走一步后对对方棋局的估计，对本方棋局的乐观估计就是对对方棋局的悲观估计，对本方棋局的悲观估计就是对对方棋局的乐观估计。因此，从下层节点向上层节点反馈信息时，乐观估计和悲观估计是交叉传递的[5]。

与极大极小算法及其改进策略不同，B*算法中的 Max 层和 Min 层都是取本层节点的最大值和最小值。算法的更新策略是，只要下层节点返回值能够使上层节点的搜索区间变小，就更新上层节点的值，更新之后上层节点的乐观值与悲观值互相靠近。下图是一棵 B*算法的搜索树更新过程的示意：

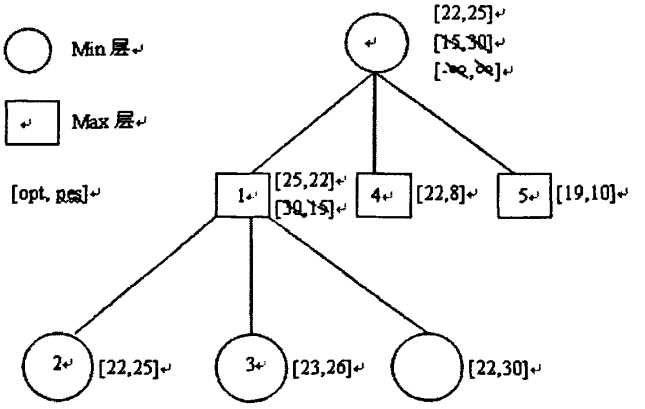


图 8. B*算法返回更新值过程

整站优化 http://www.001baidu.com/

B*算法不设定搜索深度，当找到最佳着法时停止。最佳着法的意思是，此着法的悲观值不小于其兄弟节点的乐观值，也就是如果选择此着法，棋局在最坏情况下也不差于选择了任意一个兄弟节点。在图 8 中，节点 1 的悲观值最后更新为 22，而它的兄弟节点 4 和 5 中最好的乐观值是 22，此时选择节点 1 作为最佳着法。对算法的控制是尽可能快的达到终止条件。在 B*树展开的过程中，只要子节点的估值有利于父节点估值的精化，即改动父节点的估计值。如果这种改动波及到父节点的估值，则根节点需考虑使用何种策略。

B*算法有两种搜索策略：证明最优（PROVEBEST）和排除其余（DISPROVEREST）。图 9 和图 10 分别说明了这两种策略：

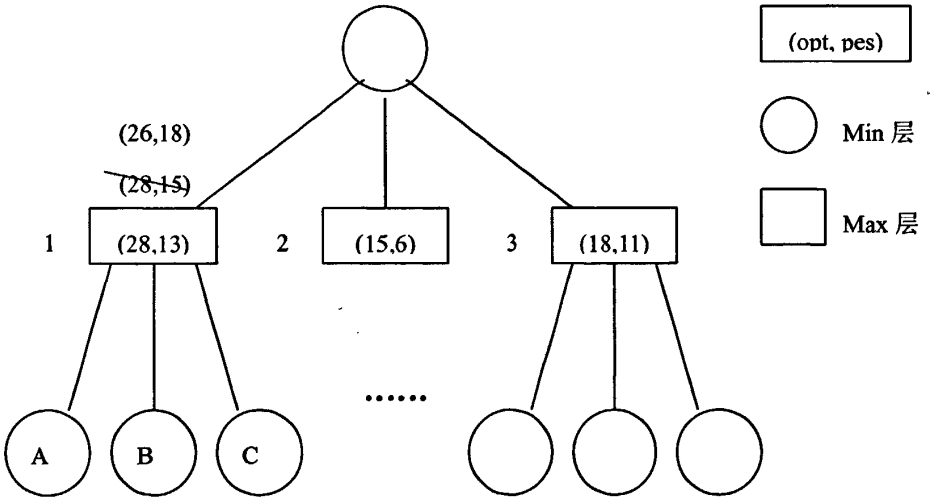


图 9. 证明最优策略（PROVEBEST）

在 Max 层中，节点 1 的乐观值最大，最佳着法最有可能出现在以节点 1 为根的子树上，因此优先对节点 1 展开搜索，返回修改乐观值和悲观值，使节点 1 的悲观值不断增加，最终大于其兄弟节点 2 和 3 的乐观值，此时证明节点 1 代表的着法最优。

整站优化 http://www.001baidu.com/

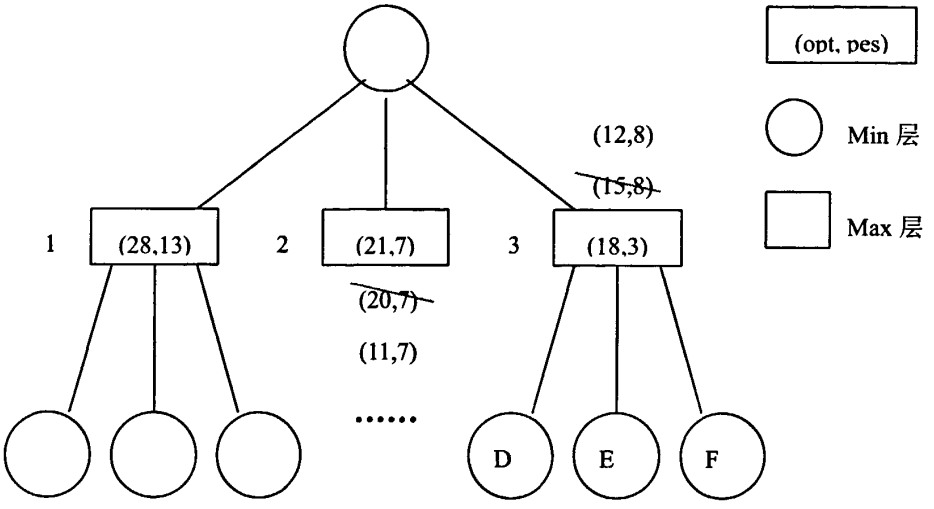


图 10. 排除其余策略（DISPROVEREST）

在 Max 层中，为了证明节点 1 最优，对节点 1 的兄弟节点展开搜索，返回修改它们的乐观值，使 2 和 3 的乐观值不断减小，最终均小于节点 1 的悲观值，此时排除 2 和 3 作为最佳着法的可能性。反证节点 1 代表的着法最佳。

在实际的应用中以 Berliner 原则来选择使用哪种策略：用一组候选分枝与最佳分枝作比较，如果各候选分枝实行信息反馈的深度是 Ti，最佳分枝实行信息反馈的深度是 t，比较 ΣTi^2 和 t^2，若前者小于后者，采用 DISPROVEREST 策略，否则采用 PROVEBEST 策略。

4.3 算法过程

下面给出 B*算法的类 C 伪代码：

首先给出要存储的节点的数据结构：

```
typedef struct BranchNode
{
    move；//当前节点代表的着法
    opt；//乐观估计
    pes；//悲观估计
    par；//父节点的地址
```


整站优化 http://www.001baidu.com/

```
    first；//第一个子节点的地址
    last；//最后一个子节点的地址
}node;
node no = {NULL，0，0，0，0，0};
node v[MAXNODE]；//预先分配内存，存储算法搜索时生成的节点
```

这一部分是算法主要过程：

```
move BSearch（void）
{
    depth = 0；//当前搜索深度
    cur = 0；//当前节点的地址
    valid = 1；//第一个可分配的地址
    v[0] = no;
    //以上为初始化过程
    while(1) //主循环
    {
        if（当前节点未被扩展过）
        {
            生成此节点下的 num 个着法
            for（每一个可行着法 m）
            {
                执行 m，创建子节点状态，存储子节点状态；
            }
            v[cur].first = valid；//修改第一个子节点位置
            v[cur].last = valid – 1 + num；修改最后一个子节点位置
            v[cur] = valid + num；//向后移动，更新可分配地址
        }
update:     for（当前节点的每一个子节点）
            {
            修改极大（小）乐观值节点位置 best，次极大（小）乐观值节点位置 next，
```


整站优化 http://www.001baidu.com/

```
        极大（小）悲观值节点位置 pest
        //depth 为偶数则取极大，为奇数则取极小
        }
        if（父节点乐观值和悲观值可以被更新）
        {
            更新相应值
            if（depth > 0）//未溯及到根节点需要回溯
            {
                cur = v[cur].par;   depth - -;   goto update; //回溯
            }
            else if（v[best].pes >= v[next].opt）//找到了最优分枝
                return v[best].move; //返回最佳着法
        }//主循环出口
        if （depth == 0）//在根节点决定下一步的策略
        {
            if（PROVEBEST）cur = best; //PROVEBEST 策略，正证最优分枝
            else cur = next; //DISPROVEREST 策略，反证次优分枝
        }
        if（depth > 0） cur = best;  depth ++; //选定策略后向更深层次搜索
    }//主循环结束
}//算法结束
```

4.4 实验结果

编码实现了 B*算法之后，将其应用于象棋小巫师 3.0[31]中，与使用了 Alpha-Beta 搜索，应用了迭代加深和历史表的程序对战，已经可以下赢采用一层搜索的程序。本程序中设计了一个简单的评估函数，以一个包含了棋子价值和位置附加值的静态评估函数给当前局面作出评价，设此局面值为 value，pes<1<opt，以 pes×value 作为悲观估计，以当前我方棋子对对方的威胁值 Thread + opt×value 作为乐观估计。其中威胁值 Thread 表示若当前局面下对方棋子在我方棋子的合法落址上（也就是我方棋子可以在下一步吃

整站优化 http://www.001baidu.com/

掉此子），那么认为我方对对方构成威胁，威胁值按照被威胁的对方棋子的价值来定，本实验中取被威胁棋子价值的一半。

下图是 B*算法执黑战胜一层搜索的程序的最后局面：

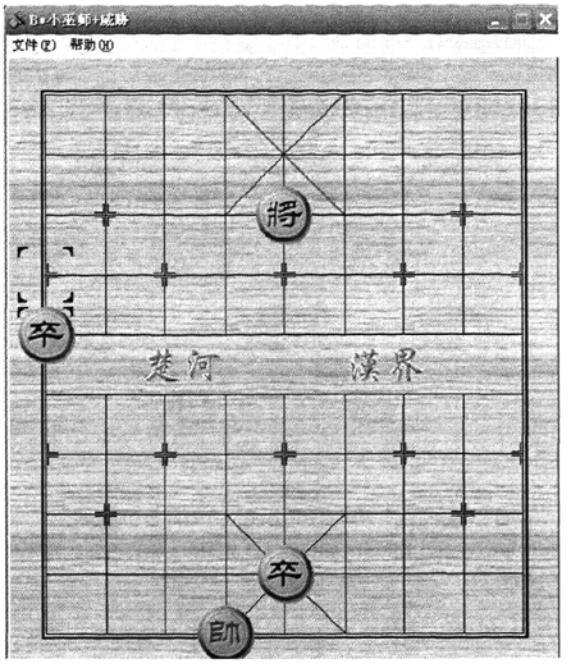


图 11. B*算法执黑战胜一层的 MAX-MIN 算法

与两层搜索的程序对战，B*算法在三十回合之后落败。

下面是 B*算法与两层搜索的程序对战时前十步的搜索时间和节点数，算法在大部分情况下搜索的节点数仅相当于一层的极大极小算法：

表 15. B*算法在前十步的搜索效率（评估函数的上下界（pes，opt）=（1，1.1））

步数	1	2	3	4	5	6	7	8	9	10
搜索节点数	41	31	34	1575	31	24	36	29	28	30
反应时间(ms)	78	78	78	115	94	94	94	94	93	78

表 16. B*算法在前十步的搜索效率（评估函数的上下界（pes，opt）=（0.9，1.1））

步数	1	2	3	4	5	6	7	8	9	10
搜索节点数	45	1814	1387	1376	36	33	18	17	23	26
反应时间(ms)	109	152	141	123	93	94	78	78	93	78

整站优化 http://www.001baidu.com/

在表 16 中，由于评估函数的上下界相差较大（系数相差 0.2），加上威胁值之后就更大了。这导致在第 2、3 步时搜索节点数要多于表 15（系数相差 0.1）在相应阶段的搜索节点数，这是因为如果某节点的初始悲观值远小于兄弟节点的乐观值时，要返回修正值使前者大于后者，就需要搜索更多的节点。

在实验中发现 B*算法具有主动进攻的特点。B*算法执红，在开局之初走出炮二进七，主动以炮换马（实验设置马的价值稍大于炮）。对方走出车 9 平 8 吃炮后，红方又以炮八平二主动打车。如果是极大极小算法执红，第一回合相同，第二回合红方就会走出炮八进七换马，也就是在前两步用炮换掉对方两个马。

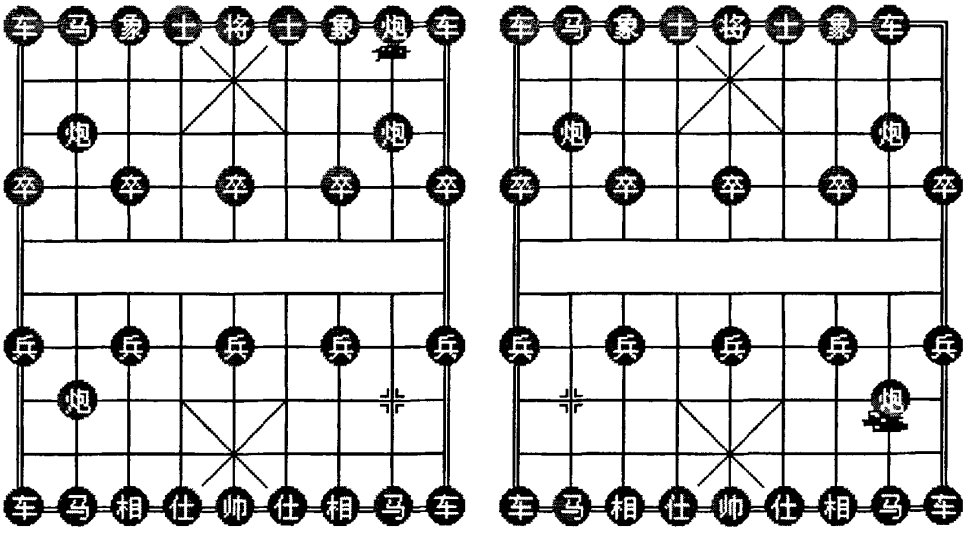


图 11. B*算法先行前两步局面

对此的解释是，B*算法由于依靠乐观值和悲观值进行搜索，之所以第二步炮八平二打车，而不是和第一步相同的以炮换马，是因为算法乐观的认为，如果对方对打车无动于衷，一个车的价值大于马的价值，因此主动打车；算法悲观的认为，即便对方对打车做出反应（实际上对方走了炮 8 平 5，从而使红炮被车捉），红方也可以躲黑车或以红车看炮（跟踪程序发现，按照悲观估值红方走出车一进二看炮），并没有损失。因此算法按照乐观估计出着。这个过程就很像是一个刚学会下棋的孩子和大人对弈一样。考虑到评估函数的设计比较简单，B*算法的棋力尚不能下赢二层搜索的算法。这个结果虽然不能令人十分满意，但也表明 B*算法可以应用到中国象棋中来，而且在能力上还有很大的提升空间。

整站优化 http://www.001baidu.com/

4.5 实验过程分析

在对 B*算法的编码实现和调试过程中，发现 B*算法存在以下问题：

1）存储空间的分配问题。

算法对存储空间要求较高，因为 B*算法需要回溯并修改上层节点的信息，这就需要将所有生成的着法存储起来作为候选着法。只要算法没有找到最佳着法，就会一直生成并存储新的着法。在实验中就需要分配较大的内存空间作为备用。

如果预先分配一块较大的内存空间，就会出现两个主要问题：一是资源浪费。因为多数情况下只需要占用少量内存算法就能正常终止，并返回最佳着法，内存分配的过大就会影响系统的其他资源请求。二是内存溢出。如果预先分配的空间过小，或是很难找到最佳着法，算法会一直占用内存资源，直到分配的空间被耗尽，此时程序出错。

动态内存分配不会浪费空间，而且符合算法的逻辑结构，也就是随生成，随分配，随占用。但是分配内存本身就需要相当一部分时间，这会延长反应时间，影响算法的效率。当算法迟迟找不到最佳着法，就需要不停的分配内存，最终会访问到保留内存，或是将所有内存占用完，操作系统出错。

2）算法的反应时间问题。

如果在某一层，假设是取最大值的那一层，所有节点的乐观值远远大于悲观值，想要达到终止条件，也就是使某个节点的悲观值不小于其所有兄弟节点的乐观值，就需要不停地搜索下层节点，逐渐增加这一节点的悲观值。这样就会占用大量的时间，使算法的反应时间延长。更有甚者，如果设计的评估函数上下界相差过大，就会出现始终无法达到终止条件的情况，最终会占满内存资源，使程序或操作系统出错。

这两个问题是算法本身固有的，无法从根本上解决。在实验当中采用了权益方法来处理这两个问题：使用的评估函数，其乐观估计值与悲观估计值比较接近，这样便于使算法达到终止条件。在调用算法之前预先分配一块大小适中的内存，当算法即将占满这块内存的时候，强行终止搜索，并返回当前的最佳着法，每当算法终止，即释放这块内存，下次调用再重新分配。

整站优化 http://www.001baidu.com/

4.6 未来工作的展望

B*算法对乐观估计和悲观估计的值很敏感，而实验中应用的评估函数非常简单，如前所述，一个静态评估函数是E，设 0<pes<1<opt, 则乐观估计值是 Thread + opt$\times E$，悲观估计值是 pes$\times E$。这种表示方式只是一种简单的模拟，是否适合 B*算法仍有待研究。相信在采用了合适的评估函数之后，B*算法的棋力会有提高。

在实验中采用了权益方法来避免算法出现问题，因此在局势较为胶着的情况下算法可能因占满内存而被强行终止，致使返回了一些次佳着法，甚至是比较短视的着法，使博弈系统的棋力大大降低，目前尚不能下赢二层搜索的 Alpha-Beta 算法。我们会不断地尝试其他的解决办法，对算法继续改进。

实验结果表明 B*算法可以应用于中国象棋博弈系统当中，但其能力仍有待提高。主要是评估函数的设计和内存的分配问题亟待解决。

整站优化 http://www.001baidu.com/

第5章 结束语

本文主要从效率上对基于剪枝策略的搜索引擎进行研究。所做的主要工作有如下几点：

1）剪枝算法的研究与实现。作为搜索引擎的核心，剪枝算法忽略了大部分冗余节点，使博弈树在一定时间内的搜索层次更深了。层数越深，返回的着法越好。比如极大极小算法在我们可接受的反应时间里只能搜索到四层，而应用了 Alpha-Beta 算法后，在同样的时间里可以搜索到七层以上。为了测试博弈能力，我们让两者对战十局，发现 Alpha-Beta 算法无一败绩。其他诸如渴望搜索和极小窗口搜索的能力也很强，甚至可以达到八层的深度。

2）辅助剪枝手段的研究与实现。迭代加深方法和历史表的引入使剪枝更早的发生，加快了程序的反应时间。着法排序的手段有很多，比如置换表、杀手启发表等等，其中存储空间要求最少的是历史启发表。使用历史启发表之后，程序的反应时间比使用之前快了将近十倍。

3）B*算法在系统中的实现。B*算法最成功的应用还要回溯到上世纪七十年代，其后对这种算法应用的研究并没有太大的进展。也很少有人关注 B*算法在中国象棋当中的适用性[30]。在 1983 年有人提出了基于概率的 B*算法，也就是 PB*算法[2]，而直到 1996 年 B*算法的提出者 Berliner 才自己对这种带有概率的 B*算法（PB*）作了详细介绍[9]。PB*引入概率因素，在发现某一分枝明显好于其他分枝时停止搜索。由于各分枝成为最佳分枝的概率不容易估计，这种 PB*算法实现起来比较困难。本文中实现了 B*算法，对于算法的效率做出了分析，应用在中国象棋当中是可行的，但在棋力上还有很大的提升空间。

通过研究和实验，我们也发现了一些问题：

1）为了使分析更有针对性，我们对各种方法都是分别单独实现的，这也导致了系统棋力不是很高。而目前流行的棋力较高的象棋软件几乎无一例外的应用了多种方法，因此可以将各种方法综合起来进行应用，想必会提高系统的实战能力。

2）B*算法的应用，虽然已经证明了是可行的，但是在实验中出现了一些问题，尤其是内存分配和评估函数的设计制约着程序能力和效率的提高。如果能够用子力价值、

整站优化 http://www.001baidu.com/

位置值和配合值作为基准，加上潜在攻击值作为乐观估计，减去受威胁值作为悲观估计，会更符合实际情况，相信也能够得到较好的实战效果。

本文中所做实验的硬件配置：CPU-P4 2.8GHz，RAM-256+256MBddr，HDD-80G；软件配置是 Windows XP SP2 版本操作系统，VC++6.0 环境。本文实现的程序是在象棋小巫师 3.0 的界面框架下完成的，在此要特别感谢黄晨对中国象棋计算机博弈所做的贡献[31]。

中国象棋计算机博弈的研究是一项长期的任务，由于中国象棋本身的复杂性，再加上作者的水平有限，在理论上和技术上都有一定的不足之处，敬请各位专家、读者批评指正。

整站优化 http://www.001baidu.com/

参考文献

[1] D. E. Knuth, R. W. Moore. An Analysis of Alpha-Beta Pruning. Artificial Intelligence 1975, 6(4): 293-326.

[2] A. J. Palay, Searching with Probabilities, Pitman Advanced Publishing Program Boston, 1985. Also, Ph.D. Thesis, CMU, 1983.

[3] Richard S. Sutton. Learning to Predict by the Methods of Temporal Differences. Machine Learning, 1988, No.3: 9-44.

[4] Bruce Abramson, Control Strategies for Two-Player Games, ACM Computing Surveys, 1989, Vol.21, No.2: 137-161.

[5] 顾立尧. 解空间搜索中的B*树算法, 上海机械学院学报, 1989, Vol.11, No.4: 59-64.

[6] 许舜钦. 电脑西洋棋和电脑象棋的回顾与前瞻.电脑学刊, 1990, 第二卷第二期: 1-8.

[7] 黄文奇, 宋恩民, 陈亮, 等. 关于象棋的不败算法. 华中理工大学学报. 1995, Vol. 32, No5: 1-4.

[8] Eyal Amir. Game Search Trees. http://citeseerx.ist.psu.edu/viewdoc/download, 1996.

[9] Berliner. H.J. and McConnell. C. B* probability based search. Artificial Intelligence, 1996, Vol.86, No.1: 97-156.

[10] 肖齐英, 王正志. 博弈树搜索与静态估值函数, 计算机应用研究, 1997, 第 4 期: 74-76.

[11] DF Beal, MC Smith. Learning Piece Values Using Temporal Differences. Journal of The International Computer Chess Association, 1997.

[12] Andreas Junghanns Are There Practical Alternatives to Alpha-Beta in Computer Chess, http://citeseerx.ist.psu.edu/viewdoc/download, 1998.

[13] TB Trinh, AS Bashi, and N Dashpande Temporal Difference Learning in Chinese Chess, Lecture Notes In Computer Science; Vol. 1416 http://citeseerx.ist.psu.edu/viewdoc/download, 1998.

[14] 陆汝钤. 人工智能. 北京: 科学出版社, 2000: 363-373.

[15] 莫建文. 机器自学习博弈策略研究与实现: 广西师范大学硕士学位论文, 2002.

[16] 王小春. PC游戏编程[M]. 重庆: 重庆大学出版社, 2002.

[17] Jaime. Carbonell. Artificial Intelligence 15-381 Heuristic Search Methods, www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/15381-f01/www/handouts/083001.ppt, 2003.

整站优化 http://www.001baidu.com/

[18] 涂志坚. 电脑象棋的设计与实现: 中山大学硕士学位论文, 2004.

[19] Shi-Jim Yen, Jr-Chang Chen, and Tai-Ning Yang. et al. Computer Chinese Chess. ICGA Journal , Taipei, Taiwan 2004: 3-18.

[20] 徐心和, 王骄. 中国象棋计算机博弈关键技术分析 J .小型微型计算机系统, 2006, 27(6): 961–965.

[21] 王骐. 博弈树搜索算法的研究及改进: 浙江大学硕士学位论文, 2006.

[22] 付强. 基于激励学习的中国象棋研究: 长沙理工大学硕士学位论文, 2006.

[23] 高强. 一种混合博弈树算法在中国象棋人机博弈中的应用研究: 大连交通大学硕士学位论文, 2006.

[24] Michael Bowling, Johannes Fürnkranz and Thore Graepel et al. Machine Learning and Games. Machine Learning, 2006, No.63: 211-215.

[25] 王赠凯, 吕维先. 机器博弈搜索技术分析. 软件导刊, 2007, Vol.2, No.1: 26-27.

[26] 魏钦刚, 王骄, 徐心和, 等. 中国象棋计算机博弈开局库研究与设计. 智能系统学报, 2007 Vol.2, No.1: 85-89.

[27] Bouke van der Spoel. Algorithms for Selective Search. http://people.cs.uu.nl. March 8th, 2007.

[28] 付强, 陈焕文. 中国象棋人机对弈的自学习方法研究. 计算机技术与发展, 2007, Vol.17, No.12: 76-79.

[29] 王洪岩, 朱峰, 张雪峰等. 一种基于神经网络的中国象棋及其博弈评估实现. 2007 中国控制与决策学术年会论文集, 2007: 687-693.

[30] 危春波. 中国象棋博弈系统的研究与实现: 昆明理工大学硕士学位论文, 2008.

[31] 黄晨等. 象棋百科全书. www.elephantbase.net.

整站优化 http://www.001baidu.com/

攻读硕士学位期间发表论文情况

[1] Xiang-Hao Pei, Qian Li, Bo Wu and Yu Lin He. An Improved Iterative Case Filtering Algorithm. Proceedings ICMLC2008, 2008, Vol.8, No.12: 4185-4190.

[2] Qian Li, Li-Hua Wang and Xiang-Hao Pei. Uncertainty Measures of Rules Based on Information Entropy and Variable Precision Rough Set. Proceedings ICMLC2008, 2008, Vol.8, No.8: 4163-4168.

整站优化 http://www.001baidu.com/

致 谢

本文是在我的导师王熙照教授和翟俊海副教授的悉心指导下完成的。王老师对我们机器博弈小组的工作非常关心，在我们进行理论研究的过程中给了我们方向性的指导。在三年的学习期间，王老师严谨的科学作风、渊博的学识给我们留下了深刻的印象，将使我们终身受益。翟老师在百忙之中抽出时间对我们的程序实现和论文编写给予了具体的建议和支持，从细节上指导我们学理论、编程序。翟老师扎实的理论功底，高尚的学术道德，对我们今后的学习和工作都影响深远。二位老师对本论文的完成倾注了大量的心血，在此表示深深的谢意。

在论文的完成期间，我们机器学习实验室的谢博鋆老师，董春茹老师，邢红杰老师以及其他所有老师都对我们提供了帮助，在此表示感谢。

还有我需要感谢的是我们博弈小组的何玉林、郭峰和邢胜同学。在三年的学习期间，我们互帮互助，情同手足，建立了深厚的友谊。在今后的工作和生活中，我们还将继续努力，共同进步。

我们机器学习实验室的同学李倩、吴博、张宁、周翠莲、王利华、赵环宇、武群夺等同学在我的日常生活和学习中也对我提供了很多有益的帮助和建议。在此一并致谢。

最后要感谢的是我的家人，他们的殷切期望和无尽关怀是我不断学习进取的动力。正是由于他们对我默默无私的支持，才使我能够顺利的完成学业。我对他们表示衷心的感谢！

整站优化 http://www.001baidu.com/

基于剪枝策略的中国象棋搜索引擎研究

万方数据 WANFANG DATA 文献链接

作者： 裴祥豪
学位授予单位： 河北大学

相似文献(10条)

1. 会议论文 褚泽永. 黄鸿. 黄远灿 浅析中国象棋计算机博弈关键技术 2006

中国象棋计算机博弈的出来的关键技术是搜索算法和评估函数.其中搜索算法大多都是从α-β剪枝算法衍生出来的.α-β剪枝算法体现了人的思维方式;同样,在评估函数的设计中采用了人工智能方法.随着计算机技术的发展,机器的计算能力再加上模拟人类思维的算法,使得在中国象棋博弈中人类最终很难战胜机器.同时,中国象棋计算机博弈的研究对人工智能的发展也起到了推动作用.

2. 学位论文 王骄 中国象棋计算机博弈关键技术研究 2006

人工智能的先驱者们曾认真地表明：如果能够掌握下棋的本质，也许就掌握了人类智能行为的核心；那些能够存在于下棋活动中的重大原则，或许就存在于其它任何需要人类智能的活动中。

计算机博弈是人工智能领域中一个重要的课题，国际象棋计算机博弈已经取得了巨大的成功，而中国象棋计算机博弈却远远落后。“棋天大圣”是东北大学人工智能与机器人研究所自主开发的中国象棋计算机博弈软件，取得了第11届世界电脑象棋奥林匹克竞赛中国象棋组的冠军。本文通过结合“棋天大圣”的研究成果，阐述了一个可以达到人类特级大师水平的中国象棋程序的设计和实现原理。

本文对中国象棋计算机博弈的关键技术进行了分析和介绍，并提出了一些新的思想和算法，以及基于数据测试集的分析和检验。其中包括以下几点。

第一，介绍了中国象棋状态空间的数据表示方法，介绍了对状态空间的数字表示和布尔表示，提出了用棋盘编码数组、棋子编码数组、映射数组数字表示状态空间，用新型数据结构路向行向比特向量与比特棋盘相结合布尔表示状态空间的新方法。在这些描述方法共同作用下，程序具有很高的运行效率，大大节省了计算时间。

第二，详细介绍了当今流行的各种着法生成算法，提出了路向行向比特向量与模板法相结合，并且辅以预置表作为辅助进行着法生成的新方法。着法生成的速度获得很大的提升。

第三，介绍了棋类搜索领域的搜索算法及其分类，给出了多种搜索算法的融合方式，以及在中国象棋计算机博弈领域应用的创新。

第四，结合“棋天大圣”的审局函数，介绍了审局函数的概念、组成和计算的方法。提出了小子同形表这种新的计算方法，可以对引擎起到良好的加速作用。

第五，本文就开局库的原理做出了介绍，提出了理想开局库并做了深入的探讨。对开局库自学习算法原理和成果进行了总结。

第六，提出了残局处理系统的新概念，将残局处理系统划分为残局知识库与残局数据库。对自行设计的残局知识库的原理、结构、实现方法做出了详细的介绍。

第七，创造性地将自适应遗传算法、神经元网络结合TD(λ)算法两种机器学习算法引入审局函数中，详细的介绍了与审局函数的结合、测试的方法以及取得的成果。本文的研究在中国象棋计算机博弈领域处于前沿，结合本文的研究成果可以创建高水平的博弈软件，而且在一系列电脑之间的比赛和人机挑战赛中，也得到了印证。本文的成果和结论，对于其它中国象棋计算机博弈程序，具有一定的参考价值。

3. 期刊论文 徐心和. 王骄. XU Xin-he. WANG Jiao 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)

机器博弈被认为是人工智能领域最具挑战性的研究方向之一.国际象棋的计算机博弈已经有了很长的历史,并且经历了一场波澜壮阔的"搏杀","深蓝"计算机的胜利也给人类留下了难以忘怀的记忆.中国象棋计算机博弈的难度绝不亚于国际象棋,不仅涉足学者太少,而且参考资料不多.在国际象棋成熟技术的基础上,结合在中国象棋机器博弈方面的多年实践,总结出一套过程建模、状态表示、着法生成、棋局评估、博弈树搜索、开局库与残局库开发、系统测试与参数优化等核心技术要点,最后提出了当前研究的热点与方向.

4. 学位论文 王涛 中国象棋计算机博弈中的增强学习研究 2006

计算机博弈曾一直被称为是人工智能研究的“果蝇”，但对于有几千年历史的中国象棋的计算机博弈的研究却远远落后于其它棋类，为了改变这种局面，东北大学成立了“棋天大圣”中国象棋计算机博弈代表队。论文选题正是来源于在队内所做的研究工作。

为了解决传统的线性评估函数对中国象棋局面的评估不够精确的问题，本文提出了两个解决方案：一个是使用TD(λ)增强学习算法优化传统的线性评估函数的可调参数；另一个是使用人工神经元BP网络替代传统的线性评估函数，然后使用TD(λ)算法训练该网络。一个中国象棋增强学习系统被设计用来实现这两个方案的学习过程。在该系统中可以使用TD(λ)算法进行四种形式的学习：专家棋谱数据库学习，自学习，固定对手学习和网络对战学习。为了验证学习的效果，设计了一个连线器用于将本系统连接到Internet上的一个网络对战平台—弈天棋缘，通过在网上擂台的等级来评价学习后的棋力提高程度。

实验表明，使用TD(λ)算法训练BP网络的方案潜力巨大，值得进行更深入的研究；使用TD(λ)算法优化线性评估函数的方案实施效果很好，可以大大提高系统的棋力。

5. 会议论文 王骄. 徐长明. 徐心和 中国象棋计算机博弈残局处理系统 2006

残局是中国象棋计算机博弈面临的一个难点,为了加强引擎在残局时的棋力,重点研究了残局处理系统,引入了残局知识库与残局数据库概念,介绍了残局知识库的入口、出口、内部实现以及对搜索的影响和在中国象棋计算机博弈中的实现问题.

6. 期刊论文 王骄. 王涛. 罗艳红. 徐心和. WANG Jiao. WANG Tao. LUO Yan-hong. XU Xin-he 中国象棋计算机博弈系统评估函数的自适应遗传算法实现 -东北大学学报（自然科学版） 2005,26(10)

使用自适应遗传算法解决中国象棋计算机博弈问题.将博弈问题分解为搜索引擎、走法生成、评估函数和开局库四大模块,然后将自适应遗传算法引入到评估函数中,通过锦标赛算法对评估函数中的参数组合进行自动调整和优化.设计并开发了基于上述方法的离线自学习系统,实验结果证明提高了程序的棋力.

7. 学位论文 何玉林 瞬时差分方法在中国象棋计算机博弈中的应用 2009

与国际象棋相比较，中国象棋具有更大的棋子运动空间，并且中国象棋的着法更加特殊、棋局变化也更加复杂。在中国象棋计算机博弈中，对于设计一款优秀的博弈软件而言，最费时的就是评价函数的实现与调整。

在本文中，我们使用三层的全连接前馈式神经网络表示评价函数，瞬时差分方法是一种增强学习算法，它利用相邻状态之间的状态值之差逐步地更新值函数的权值。基于单输出的神经网络，通过将TD（λ）算法引入到中国象棋计算机博弈中，我们推导出了一套新的神经网络权值更新规则。我们使用专家棋谱作为训练样例，并利用新权值更新规则完成对网络的更新。

在训练神经网络的过程中，每份专家棋谱被重复地学习，直至评价函数对专家棋谱中每个局面给出的评价值达到稳定。通过实验验证，我们学习到的评价函数是可行有效的。并且，我们获得了评价函数的表现与三种参数之间的关系：评价函数表现与神经网络隐含层节点个数之间的关系、评价函数表现与学习率α之间的关系、以及评价函数表现与参数λ之间的关系。

8. 会议论文 王骄. 王愿博. 王晓鹏 中国象棋计算机博弈着法生成研究 2006

着法是博弈树展开的依据,着法生成的效率直接决定了博弈树的展开速度,对搜索引擎的效率影响巨大.本文就中国象棋计算机博弈中常见的着法生成

整站优化 http://www.001baidu.com/