重庆大学

硕士学位论文

# 六子棋计算机博弈及其系统的研究与实现

姓名：李果

申请学位级别：硕士

专业：模式识别与智能系统

指导教师：李祖枢

20070428

# 摘　要

计算机博弈是人工智能领域一个极其重要且最具挑战性的研究方向之一，它的研究为人工智能带来了很多重要的方法和理论，产生了广泛的社会影响和学术影响以及大量的研究成果。在过去的半个世纪里,世界各地的学者花费了大量的心血对于计算机博弈包括奥赛罗、checker、国际象棋、中国象棋、五子棋、围棋进行研究。这是因为计算机博弈是人工智能的一块试金石，然而棋类游戏又是计算机博弈的一个标准性问题，各种搜索算法、模式识别及智能方法在计算机博弈中都可以得到广泛的应用。在长时间的研究中,涌现出大量令人震惊的成果,1997 年"深蓝"战胜卡斯帕罗夫的比赛就在全世界范围内引发了震动。其他很多棋类的计算机水平都已达到了世界冠军的水平。

目前，对于像五子棋、中国象棋等棋类游戏的计算机博弈算法研究已相对成熟，六子棋作为一个刚刚兴起不久的棋类游戏，其计算机博弈算法的研究还相对较少。即使目前已经出现六子棋的论坛以及比赛的平台，但只限于人人对弈。真正对于六子棋计算机博弈算法以及系统的研究还不多。六子棋的发明者台湾吴毅成教授给出了六子棋的公平性问题以及基于追著(Threats)的胜利策略，但是对于其计算机博弈问题没有给出更加深刻的阐述，同时也没有全面解决六子棋计算机博弈问题。本文正是对六子棋计算机博弈技术的一个探索。

本文中提出的六子棋计算机博弈系统可以分为四个主要部分：搜索引擎、走法生成、评估函数和开局库。搜索引擎模块包含了比较成熟的搜索算法，以及对他们的结合和优化；走法生成模块是对搜索的结果进行比较处理，确定当前的走法；评估函数模块中本文根据棋型特征构建了着子棋力的评估函数并提出用遗传算法来做评估函数参数的调整与优化的方法；开局库存储了大量的专家棋谱，可以避免在开局时由于搜索深度的不足而带来战略上的失误，同时大大提高了对战的效率。

最后本文对六子棋计算机博弈系统进行了测试与评价，包括评估函数的准确度、搜索算法的效率以及系统的整体性能。

**关键词：六子棋计算机博弈，博弈树，评估函数，锦标赛算法，遗传算法**

# ABSTRACT

Computer game is one of the most important and challenge subject of the AI field, which brings many important theories and methods for the research of AI and achieves so many contributions that wildly influences the society and academic. In the past half century time scholars from all over the world poured numerous time and mind into the research of computer game including Othello、checker、chess、Chinese chess、Go-Moku and weichi, because computer game is the touchstone in the AI field and the chess game is one of the standard problem of the computer game which contains the applications of all kinds of search algorithm , mode identification and intelligence method. A good many of shocking results are achieved in the long lasting researching, for example the world chess game champion Kasparov lost his match with the super computer "Deep Blue" which gave the world a big shock in 1997.Now the AI of many other chess games has reached to the level of world champion.

Comparing to the mature computer game algorithm of Go-Moku and Chinese chess now, there are few researches on connect6 which is recently arising. Although the connect6 game is now appear in some forum and matches on the Internet, its players are just people. Professor Yicheng Wu the inventor of the connect6 in Taiwan has given the fairness problem's solution and winning strategy based on threats, but there is no more deep research of the computer game problem of connect6,neither a completely solution of the computer game problem of connect6 is given. This paper makes right a study on the computer game of connect6.

The computer game system of connect6 given in this paper can be summed to four parts: searching engine, move generator, evaluation function and starting steps database. The searching engine contains mature searching algorithms which are combined and optimized. The move generator compares the searching result and chose the best for the next step. The evaluation function is constructed according to the characteristic of chess type and putting forward the method that making use of genetic algorithm to adjust and optimize the parameters. A lot of experts' chess examples are stored in the starting steps database to avoid the strategy mistakes caused by the insufficiency of searching depth, which also brings lots of efficiency in actual playing.

As conclusion the testing and valuing of the computer game system of connect6 is given at last, include the precision of the evaluation function, the efficiency of the searching algorithm and the efficiency of the whole system.

**Keywords:** Computer Game Of Connect6, Game Tree, Evaluation Function, Tournament Algorithm, Genetic Algorithm

# 独 创 性 声 明

本人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。据我所知，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过的研究成果，也不包含为获得 重庆大学 或其他教育机构的学位或证书而使用过的材料。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谢意。

学位论文作者签名：李果　　签字日期：2007年6月6日

# 学位论文版权使用授权书

本学位论文作者完全了解 重庆大学 有关保留、使用学位论文的规定，有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许论文被查阅和借阅。本人授权 重庆大学 可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

保密（　），在____年解密后适用本授权书。

本学位论文属于

不保密（√）。

（请只在上述一个括号内打“√”）

学位论文作者签名：李果　　导师签名：

签字日期：2007年6月6日　　签字日期：2007年6月7日

# 1 绪 论

[本章摘要]本章简述了目前计算机博弈技术的发展状况，通过把目前刚刚兴起不久的棋类游戏——六子棋作为计算机博弈技术的研究平台，提出了六子棋计算机博弈的概念。通过分析六子棋计算机博弈技术在整个计算机博弈技术中应用的地位，明确了进行本课题研究的重要性与必要性。然后通过对国内外相关研究现状的综述，指出本课题研究的意义。最后提出了本文所要研究的主要课题并介绍了所完成的工作。

## 1.1 引言

人工智能诞生50周年以来，在知识工程、模式识别、机器学习、进化计算、专家系统、自然语言处理、数据挖掘、机器人、图象识别、人工生命、分布式人工智能等各个领域得到了蓬勃的发展。而计算机博弈作为人工智能领域的一个重要分支，也得到了极其快速的发展，并为人工智能带来了很多重要的方法和理论，同时也产生了广泛的社会影响和学术影响以及大量的研究成果。代表计算机博弈技术的各种棋类游戏在其各自的计算机博弈技术研究中已经取得了相当丰硕的成果，且其计算机博弈系统也日趋完善，基本上能达到大师级水平。六子棋作为最近两年才兴起的棋类游戏，其计算机博弈技术和算法的研究相对较少。本文提出了对六子棋计算机博弈及其系统的研究与实现，也正是对六子棋计算机博弈技术的一个探索。[1] [2] [3]

## 1.2 六子棋计算机博弈的研究意义

计算机博弈又称机器博弈，而机器博弈是博弈论应用的一个方面。那么什么是博弈论？首先谈谈什么是博弈。博弈是指一些个人、团队或其他组织，面对一定的环境条件，在一定的规则约束下，依靠所掌握的信息，同时或先后，一次或多次，从各自允许选择的行为或策略进行选择并加以实施，并从中各自取得相应结果或收益的过程。博弈论又称对策论，就是系统研究各种各样博弈中参与人的合理选择及其均衡的理论。博弈论广泛地应用于经济、军事、航空调度、天气预报、资源勘探等各个方面。当然，计算机博弈也是其应用之一，同时，计算机博弈又是人工智能的经典研究领域。那么计算机博弈就把计算机（机器）、博弈论、人工智能三者组成了一个有机整体，对于计算机博弈的研究，既能推动博弈论的发展，又能推动人工智能的发展。[4] [5]

六子棋由台湾吴毅成教授于两年前发明，现在逐渐开始兴起。在这期间，

已经有好几个站点支持在线的六子棋游戏。比较著名的有以下几个站点：

CYC game site: http://www.cycgame.com/. (Many good Chinese players in this site)

Little Golem: www.littlegolem.net (many European good players in this site)

pente.org

www.ludoteka.com (Spanish; a Spanish introduction about Connect6 can be found in gomezarrausi.blogspot.com.)

BrainKing.com.

其中，以 CYC game site 和 Little Golem（简称 LG）最为著名，许多优秀的中外选手在这里留下了他们的辉煌战绩。同时，也出现了相关的六子棋论坛：

http://www.connect6.org/forum (English)

http://www.connect6.org/chinese/forum (Traditional Chinese)

http://bbs.cycgame.com/ (Traditional Chinese)

http://groups.msn.com/connect6 (Traditional Chinese)

http://connect6.baidubar.com/bar/ (Simplified Chinese)

等等，热爱六子棋的人们活跃在这些论坛上，经常给我们一些精彩的对局以及关于六子棋的讨论。[6]

尽管六子棋游戏已经兴起，并且出现了人人对弈的站点和相关的论坛，但真正把六子棋引入到学术上来研究的事实几乎为零。除了吴毅成教授给出了六子棋的公平性问题以及基于迫著(Threats)的胜利策略等外，六子棋计算机博弈问题很少被提及。六子棋计算机博弈系统也只有 NCTU6、X6 和 EVG，但都基本采用了吴毅成教授所给出的基于迫著(Threats)的胜利策略，并没有把计算机博弈技术完全地引入到计算机博弈系统的研究上来。[7]

世界各国的学者已经成功地将计算机博弈技术引入到国际象棋、中国象棋、五子棋、围棋等棋类中，很多博弈算法已相当成熟，其计算机的水平也基本达到或超过了世界冠军的水平。但对于不同的棋类，其计算机的博弈算法有其各自的特点，计算机博弈系统的实现方法也是不一样的。

六子棋作为一个才刚刚兴起的棋类游戏，由于它的特殊性，每次每方走下两颗棋子。直观地看，其状态空间复杂度以及博弈树的复杂度会成倍地提高，而且必须考虑两步棋的综合效用，也就是综合评估，在搜索算法上也必须做到“两步”当“一步”处理，也就是综合搜索。在此，我们提出六子棋计算机博弈，以六子棋作为平台，来研究计算机博弈技术。六子棋计算机博弈和其他棋类的计算机博弈一样，都会用到一些最基本的搜索算法、模式识别及智能方法，并且可以采用一些发展较为成熟的计算机博弈技术。但由于六子棋的特殊性，上文已经提到，又不得不对已有的计算机博弈技术做出相应的改进，来适应六子

棋计算机博弈的要求。这也正体现了六子棋计算机博弈的一般性和特殊性。一般性：六子棋将会采用已有的成熟的计算机博弈技术；特殊性：六子棋根据自身的特点，对通用的计算机博弈技术做出改进，来适应自身的发展要求。

对六子棋计算机博弈的研究，不但能促进六子棋这项运动的发展，而且更能进一步推动计算机博弈理论的发展，甚至博弈论、人工智能的应用与发展。把六子棋人人对弈的局面转到可以人机大战上来，并且这对宽带娱乐、棋类教学也是非常有意义和帮助的。

## 1.3 六子棋计算机博弈在国内外的研究现状

### 1.3.1 计算机博弈研究简史[8] [9] [10] [11] [12] [13] [14] [15]

计算机博弈，简单的说，就是让计算机像人一样从事需要高度智能的博弈活动。研究者们从事研究的计算机博弈项目主要有国际象棋、围棋、中国象棋、五子棋、西洋跳棋、桥牌、麻将、Othello、Hearts、Backgammon，Scrabble 等。其中二人零和完备信息博弈的技术性和复杂性较强，是人们研究博弈的集中点。二人零和随机性研究的一个代表是 Backgammon，并且产生了很大影响。桥牌是研究不完备信息下的推理的好方法。高随机性的博弈项目趣味性很强，常用于娱乐和赌博，是研究对策论和决策的好例子。

近代计算机博弈的研究是从四十年代后期开始的，国际象棋是影响最大、研究时间最长、投入研究精力最多的博弈项目，成为计算机博弈发展的主线。

1950 年 C.Shannon 发表了两篇有关计算机博弈的奠基性文章(Programming A Computer for Playing Chess 和 A Chess-playing Machine)。1951 年 A. Turing 完成了一个叫做 Turochamp 的国际象棋程序，但这个程序还不能在已有的计算机上运行。1956 年 Los Alamos 实验室的研究小组研制了一个真正能够在 MANIAC-I 机器上运行的程序（不过这个程序对棋盘、棋子、规则都进行了简化）。1957 年 Bernstein 利用深度优先搜索策略，每层选七种走法展开对局树，搜索四层，他的程序在 IBM704 机器上操作，能在标准棋盘上下出合理的着法，是第一个完整的计算机国际象棋程序。

1958 年，人工智能届的代表人物 H.A.Simon 预言："计算机将在十年内赢得国际象棋比赛的世界冠军。"当然，这个预言过分乐观了。

1967 年 MIT 的 Greenblatt 等人在 PDP-6 机器上，利用软件工具开发的 Mac Hack VI 程序，参加麻省国际象棋锦标赛，写下了计算机正式击败人脑的记录。

从 1970 年起，ACM Association for Computing Machinery 开始举办每年一度的全美计算机国际象棋大赛。从 1974 年起，三年一度的世界计算机国际象棋大

赛开始举办。

1981 年, CRA YBL ITZ 新的超级计算机拥有特殊的集成电路, 预言可以在 1995 年击败世界棋王。1983 年, Ken Thompson 开发了国际象棋硬件 BELL E, 达到了大师水平。

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"1987 年,"深思"首次以每秒 75 万步的思考速度露面, 它的水平相当于拥有国际等级分为 2450 的棋手..1988 年,"深思"击败丹麦特级大师拉尔森。1989 年,"深思"已经有 6 台信息处理器, 每秒思考速度达 200 万步, 但在与世界棋王卡斯帕罗夫进行的"人机大战"中, 以 0 比 2 败北。

1997 年，由 1 名国际特级大师, 4 名电脑专家组成的"深蓝"小组研究开发出"更深的蓝"，它具有更加高级的"大脑"，通过安装在 RS6000S 大型计算机上的 256个专用处理芯片, 可以在每秒钟计算2 亿步, 并且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 击败了卡斯帕罗夫。成为人工智能领域的一个里程碑。

在其它博弈项目上, 1962 年 Samuel 等人利用对策理论和启发式搜索技术编制的西洋跳棋程序战胜了美国的州冠军。1979 年 H.Berliner 的程序 BKG9.8 以 7 比 1 战胜了 Backgammon 游戏的世界冠军 Luigi Villa。1980 年美国西北大 Mike Reeve 的程序 The MOOR 战胜了 Othello 世界冠军。

1989 年第一届计算机奥林匹克大赛在英国伦敦正式揭幕，计算机博弈在世界上的影响日益广泛。

### 1.3.2 六子棋计算机博弈的研究现状[7]

由于六子棋才发明不久，目前在学术界引起的关注还是很少，国内也只有台湾的一些大学和我们在研究，国外几乎没有。但是民间对六子棋的兴趣很高，由于它的简单规则但具有很高的复杂度。网上的交流也很热闹。

六子棋的发明者台湾的吴毅成教授所带领的六子棋研究小组设计并开发了名为 NCTU6（交大六号）的六子棋计算机博弈程序。它采用的是 α-β 搜索树，深度为 3，以及结合基于双迫著的迫著空间搜索。目前，吴和 chang 合作正在开发一个新的程序，该程序打算把 single-threat 引入到 threat-space 搜索中。

迫著(Threats)的定义如下：对于六子棋，假设一个玩家（称为白，W）不能连 6。如果 W 需要下 t 颗子来避免 B 连成六子，则称 B（黑）有 t 个迫著。例如下图的(a) 单迫著(one threat), (b) 双迫著(two threats), (c) 三迫著(three threats)。

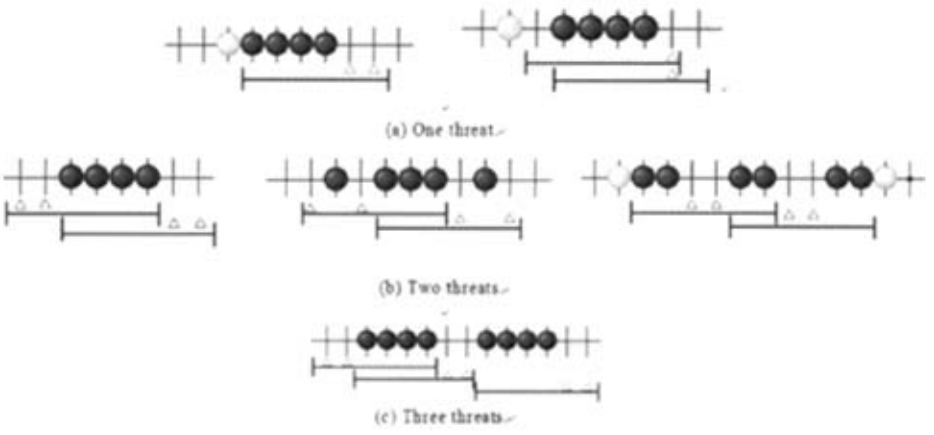


图 1.1 迫著示意图
Figure1.1 threats

如果一次下子后会产生一个迫著，则该下子称为 single-threat move。产生两个迫著，则该下子称为 double-threats move。如果是三迫著的情况，则 B 赢得比赛。而对六子棋来说，赢的策略就是阻挡所有对方的迫著，并同时产生三个或以上的迫著。

另外，已证明每次下一颗子，最多可产生两个迫著如下理论。

这理论衍生出活三、死三、活二、死二定义：若仅再下(4-t)颗子，就可以产生一个迫著，则为死-t 迫著；若仅再下(4-t)颗子，就可以产生两个迫著，则为活-t 迫著。我们可以从以下的图看出。

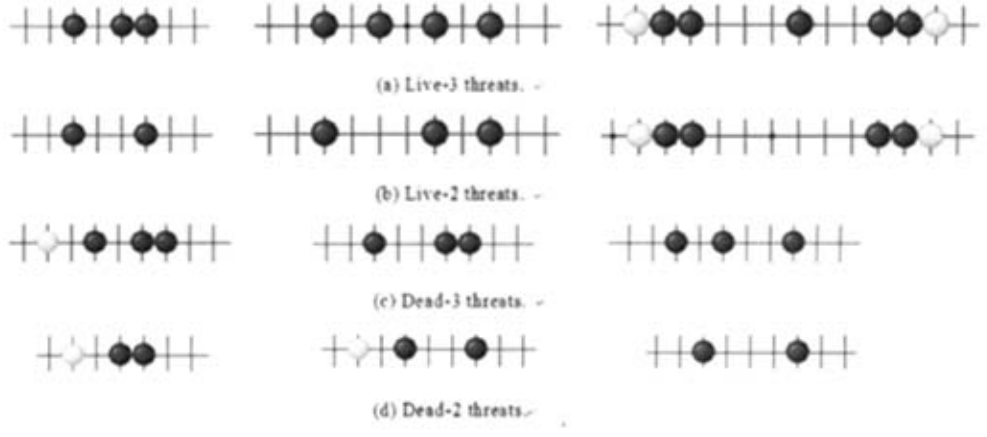


图 1.2 -t 迫著
Figure1.2 –t threats

对六子棋而言，活三、死三、活二、死二都很重要，这是因为每次可下两颗子，所以下一手棋就有机会形成真正的迫著(threats)。在所有的四种迫著中，

活三、活二、死三迫著也被称为高度潜在迫著或 HP-THREATS,因为一个子可以创造至少一个迫著或者两个子可以创造至少两个迫著。在所有的 HP-threats 中,活三和死三也被称为 HP3-threats，该迫著只需要下一个子就可以有至少一个迫著。玩家在进攻时通常想结合真正的迫著和 HP-threats。这是在六子棋中十分有效的策略。

另外，现有的六子棋计算机博弈程序还有前台大教授许舜钦之团队的 EVG 和刘思源及颜士净的 X6，他们也基本上采用了上述策略。

## 1.4 六子棋计算机博弈的核心问题

六子棋计算机博弈系统包括四个主要部分：搜索引擎、走法生成、评估函数和开局库。在搜索引擎模块中采用什么样的搜索算法，在走法生成模块中怎样确定走法，在评估函数模块中如何确定参数以及怎样确定适应度函数，在开局库中存储哪些开局模板，这都是我们研究六子棋计算机博弈需要解决的问题。其中，搜索算法和评估函数是六子棋计算机博弈中我们需要深入研究和解决的核心问题。[16]

### 1.4.1 搜索算法

所有棋类的计算机博弈都涉及到搜索算法，搜索算法就是要根据当前的棋局状态以及规定的搜索深度和宽度，在博弈树中找到一条最佳路径。六子棋计算机博弈系统中的搜索引擎模块需要完成以下几部分的工作：确定采用什么样的搜索算法、加入关于搜索的启发式信息、尽可能缩小博弈树的规模而避免冗余计算。这是其他棋类计算机博弈也需考虑的问题。

而由于六子棋的特殊性，每次每方走两颗棋子，又必须考虑两步棋的综合效用。而通常的搜索算法必须把两步当作一步处理，如何把“两步”变为“一步”，这也是我们需要研究和解决的问题，也就是一个综合搜索的问题。

### 1.4.2 评估函数

评估函数是模式识别和智能算法应用最为广泛的领域。不管多么复杂的评估函数，都可以表示为一个多项式。评估函数一般来说必须包括 5 个方面的要素，分别是固定子力值、棋子位置值、棋子灵活度值、威胁与保护值、动态调整值，每一方面的值又是由许多参数值构成的。即使最简单的评估函数也有 20 多个参数，将这些值线性地组合在一起得到最终的评估值。[16]

六子棋计算机博弈系统中评估函数模块的任务是：确定评估函数的参数类型以及参数个数，研究和解决“两步”到“一步”的综合评估问题以及对评估函数的优化问题。例如我们提出了用遗传算法对评估函数参数进行调整与优化的方法。

## 1.5 六子棋计算机博弈系统的评价方法

六子棋计算机博弈系统需要一个评价机制来衡量系统的性能。我们从以下三方面给出系统性能的评价指标：

第一、评估函数准确度

第二、搜索算法的效率

第三、系统整体性能

有了这些评价指标后，需要用已有的六子棋计算机博弈系统（比如 NCTU6）或用户玩家作为本系统的对手，作机机对弈或人机对弈，通过比赛的结果来测评本系统的性能。

## 1.6 课题的提出和研究意义

### 1.6.1 课题的提出

计算机博弈技术发展到今天，已经为人工智能带来了很多重要的方法和理论，并且产生了广泛的社会影响和学术影响以及大量的研究成果。一些特殊的技术被应用到计算机博弈系统中，例如基于规则的人机对弈系统、博弈树并行搜索算法、分布式博弈搜索算法、基于 TD 强化学习的智能博弈等等。

六子棋虽然刚刚兴起，但在民间已悄然流行起来。六子棋人人对弈的平台以及论坛已经建立起来，人们对六子棋的关注也越来越多。但对于六子棋计算机博弈的研究还相对较少，因此我们提出六子棋计算机博弈以及对其系统的设计与开发。

与五子棋、国际象棋、中国象棋、围棋等其他棋类不同，六子棋每次每方走两颗棋子，这就决定了在六子棋计算机博弈系统中必须采用与其他棋类不同的博弈技术和方法。我们所提出的六子棋计算机博弈系统可以分为四个主要部分：搜索引擎、走法生成、评估函数和开局库。在每个模块中采用一些针对六子棋特殊性的技术和方法，来解决六子棋计算机博弈问题。

### 1.6.2 研究的意义

我们将通过对六子棋计算机博弈及其系统的研究与实现，建立一个六子棋的人机对弈平台，真正实现六子棋的人机对战时代，并且希望计算机博弈这一人工智能的重要分支领域取得新的突破，同时也希望能对博弈论的发展起到积极的推动作用。而且对于宽带娱乐、棋类教学也是非常有现实意义的。

## 1.7 本文的主要研究内容

本文已经把将要设计与开发的六子棋计算机博弈系统分为四个主要部分：搜索引擎、走法生成、评估函数和开局库。而本文主要进行对搜索引擎、走法

生成、评估函数这三部分的研究。

在搜索引擎模块部分，将采用带有启发式信息的 α-β 剪枝搜索算法；在走法生成模块部分，重点解决六子棋中“两步”到“一步”的映射问题；在评估函数模块中，针对棋型特征构建着子棋力的评估函数并提出用遗传算法来对评估函数参数进行调整与优化的方法。

# 2 六子棋平台介绍[7] [17] [18]

[本章摘要]本章对六子棋以及其平台的相关概念作了基本的介绍，并对六子棋的特点作了简要的分析，同时也关注着六子棋的发展动向。

## 2.1 背景

台湾国立交通大学资讯工程系吴毅成教授在最近两年发展出一系列 K 子棋，其中最有趣的就是六子棋（或连六棋），其英文名是 Connect6。

K 子棋被定义为：Connect（m, n, k, p, q）

m, n —— 大小为 m×n 的棋盘；

k —— 某一方首先在横向、竖向、斜向任一方向上形成连续 k 颗自己的棋子便可获取游戏的胜利；

q —— 比赛开始时第一方（默认为黑方）落下 q 颗棋子；

p —— 此后黑白双方每次落下 p 颗棋子

那么，Connect（k, p, q）表示 Connect（∞,∞, k, p, q），也就是在一个无限大的棋盘上的 K 子棋。传统的五子棋就可以表示为 Connect（15,15,5,1,1）。我们现在所说的六子棋就可以表示为 Connect（m,n,6,2,1），吴教授所推荐的适合竞赛的棋盘大小为 19×19 或 59×59，我们选择前一种棋盘大小，所以可以表示为 Connect（19,19,6,2,1），在后文所提及的六子棋都是指此定义，除非特殊情况，不再作说明。

## 2.2 规则

六子棋的规则与传统的五子棋（指没有禁手的五子棋）非常相似，规则非常简单仅有以下三条：

玩家：如五子棋和围棋，有黑白两方，双方各执黑子和白子，黑棋先行。

规则：六子棋的玩法是除了第一手黑方先下一颗子外，之后黑白双方轮流每次各下两子，在横向、竖向、斜向先连成连续的六子或六子以上的一方为赢家。

若全部棋盘填满仍未分出胜负，则为和棋。

没有禁手，例如长连仍算赢。

注：连珠棋规则或国际五子棋规则，有其他额外的禁手及开局规则。而六子棋则完全不需要！

棋盘：因公平性不是问题，所以棋盘可以无限大。然而为了让游戏可实际地来

玩，目前棋盘都采用围棋的十九路棋盘。

## 2.3 公平性问题

我们首先来简单介绍一下五子棋的公平性问题，然后根据公平性的一些定义引出六子棋的公平性问题。

### 2.3.1 五子棋的公平性问题

五子棋一般规则的不公平理由很简单：每当黑方下出一步后，比白方盘面多一颗子；然而每当白方下出一步后，盘面子数却只能与黑方打平。最近几年，有计算机专家已经证明出先下必胜的结论。而远在1903年日本棋院就限制双三、双四、长连等禁着，并称之为连珠棋(Renju)，专业棋士仍然认为对黑有利，计算机专家后来也又证明连珠棋仍然是先下必胜。

在学界，Allis 等人(Allis 1994; Allis, Herik and Huntjens, 1995)是第一个证明出一般规则黑必胜。

1998年国际五子棋协会(Renju International Federation: RIF)发展了新的五子棋国际规则，限制许多开局的下法，来更进一步限制黑方的优势。但对顶尖专业棋士或程序而言，公平性的要求是相当高的。若某些棋型被证明出必胜或必败，对顶尖专业棋士或程序就少了一些变化。国际连珠棋专家也了解这问题，也对这问题有一些相关的讨论。 最近 2003 年 RIF 又继续提出要征求新的五子棋国际规则。至于，对未来的国际连珠棋，相信会有不错的改良，但目前我们无法评论。

过去五子棋的公平性问题，也产生了一个副作用：那就是让棋盘变小。Sakata 及 Ikawa 两位提到愈大的棋盘，愈增加黑方获胜的机会，因此需要缩小棋盘大小，这就是现有五子棋 15x15 的棋盘。然而很矛盾的是小棋盘反而让计算机更容易算出五子棋的胜负。

### 2.3.2 公平的定义

Van den Herik、Uiterwijk、Van Rijswijck 等人于 2002 年，给了"公平"一个适当的定义如下：若该游戏是平手的游戏，且双方犯错机率是相等的话，则可称此游戏是公平的。然而，"双方犯错机率是相等"的数学模式很难建立；这是因为若有新的下棋策略被发明后，则犯错机率算法就会不同，就会影响公平性。因此，很难用建立数学模式来证明公平。反过来，要证明不公平则比较容易且可行的。

以下是吴毅成教授给出的定义：

定义一：明确不公平性(definite unfairness)：若已经证明出一方必胜，则此游戏可称为明确不公平。例如：用一般规则的五子棋为明确不公平的。

定义二：单调不公平性(monotonical unfairness)：若已经证明出一方必然不会必

胜，但尚无法证明另一方必然不会必胜，则此游戏可称为单调不公平。例如：K 子棋中 Connect(m,n,k,p,p)，可用策略盗用论点(Strategy-stealing arguments)，证明白方必然不会必胜；因此，Connect(6,1,1),Connect(7,1,1),Connect(6,2,2)等皆为单调不公平的。然而，因为 Connect(8,1,1)已被证明双方平手，所以不是单调不公平的。

定义三：经验上不公平性(empirical unfairness)：若大多数棋士尤其是专业棋士经过实际的下棋经验认定一方必胜或有极高胜率，则此游戏可称为经验上不公平。例如：在早期用一般规则的五子棋及连珠棋，已被一般棋士认定是黑方必胜；因此在当时可称为经验上不公平。

定义四：潜在公平性(potential fairness)：若该游戏尚未被证明出或论证为明确不公平、单调不公平、经验上不公平的话，则此游戏可称为潜在公平。

依据此定义，一个目前为潜在公平的游戏，不见得能持续在未来仍为潜在公平；一个游戏能持续为潜在公平愈久，则成为公平的机会就愈高。

### 2.3.3 脱离战场

脱离战场是下棋的战略之一，是指将棋下在远离战场的一端。初始脱离战场是指白方的第一手棋（黑方的第一手之后）远离黑方的第一子。

若初始脱离战场没有对白棋造成不利的后果，则当黑棋去挡这些白棋后，会成为类似 Connect(6,2,2)；很明显，这会形成为单调不公平。

### 2.3.4 六子棋的公平性问题

对六子棋来说，每当一方下出一步（两子）时，该方一定比对方多出一颗子。直观上，这很自然地使得六子棋具有相当的公平性；当然如上所言，目前仍然不能依此论证六子棋是绝对公平的。但是，至少可以依据以下论点，来论证六子棋目前仍是潜在公平的：

① 目前，尚无人能证明六子棋是明确不公平。

② 目前，尚无人能证明六子棋是单调不公平。另外，吴毅成教授已在他们的论文中证明白方不能采用初始脱离战场策略，否则黑方胜。这个理论暗示双方必须从中心点开始缠斗；且我们不能从上述初始脱离战场理论，推论出单调不公平性。

③ 目前，尚无人能证明六子棋是经验上不公平。目前，已有许多六子棋好手研究许多定石及诘棋，尚无人能认定对某方有利。例如：诘棋一中的白 6&7，本来认定是白必胜，但目前发现其实还有待更深入的研究。

当然，此游戏的公平性，确实需要来获得更多的证据和长时间来验证。

## 2.4 复杂度

对六子棋而言，因为公平性不是问题，所以棋盘是可以任意地大，甚至是无限大亦可。state-space 复杂度可达 $10^{172}$，与围棋相当。game-tree 的复杂度可达 $10^{140}$（按 30 手计算），远大于五子棋，与象棋相当。目前，由于许多高段棋士对此游戏的逐渐了解，常会下到 40 多手，若以此推算也可达到 $10^{188}$，这个复杂度已是远大于象棋的复杂度了。这里给出了一些常见棋类游戏的 state-space 复杂度和 game-tree 的复杂度，如下图所示：

State-space complexities and game-tree complexities of various games

| Id. | Game | State-space compl. | Game-tree compl. | Reference |
|---|---|---|---|---|
| 1 | Awari | $10^{12}$ | $10^{32}$ | [3,7] |
| 2 | Checkers | $10^{21}$ | $10^{31}$ | [7,94] |
| 3 | Chess | $10^{46}$ | $10^{123}$ | [7,29] |
| 4 | Chinese Chess | $10^{48}$ | $10^{150}$ | [7,113] |
| 5 | Connect-Four | $10^{14}$ | $10^{21}$ | [2,7] |
| 6 | Dakon-6 | $10^{15}$ | $10^{33}$ | [62] |
| 7 | Domineering (8 × 8) | $10^{15}$ | $10^{27}$ | [20] |
| 8 | Draughts | $10^{30}$ | $10^{54}$ | [7] |
| 9 | Go (19 × 19) | $10^{172}$ | $10^{360}$ | [7] |
| 10 | Go-Moku (15 × 15) | $10^{105}$ | $10^{70}$ | [7] |
| 11 | Hex (11 × 11) | $10^{57}$ | $10^{98}$ | [90] |
| 12 | Kalah(6,4) | $10^{13}$ | $10^{18}$ | [62] |
| 13 | Nine Men's Morris | $10^{10}$ | $10^{50}$ | [7,44] |
| 14 | Othello | $10^{28}$ | $10^{58}$ | [7] |
| 15 | Pentominoes | $10^{12}$ | $10^{18}$ | [85] |
| 16 | Qubic | $10^{30}$ | $10^{34}$ | [7] |
| 17 | Renju (15 × 15) | $10^{105}$ | $10^{70}$ | [7] |
| 18 | Shogi | $10^{71}$ | $10^{226}$ | [76] |
| | (Source: [Herik et al 2002] [Wu et al 2006]) | | | |
| | Connect6 | $10^{172}$ | $10^{140}$-$10^{188}$ | |

图 2.1 状态空间和博弈树复杂度

Figure2.1 state-space and game-tree complexity

## 2.5 六子棋定石

定石（Joseki）是指经过长期并完整地研究出较为固定的开局方式，英文通常称之为 Joseki（来自日语）。以下是最常见的 20 种开局方式：

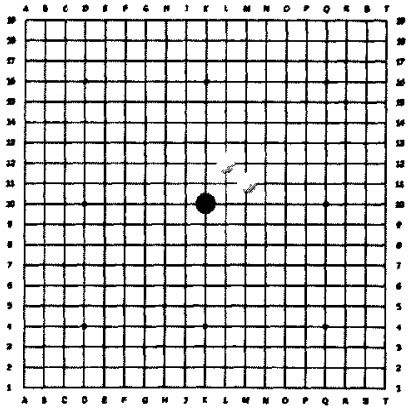

图 2.2 日日局
Figure2.2 ri-ri start

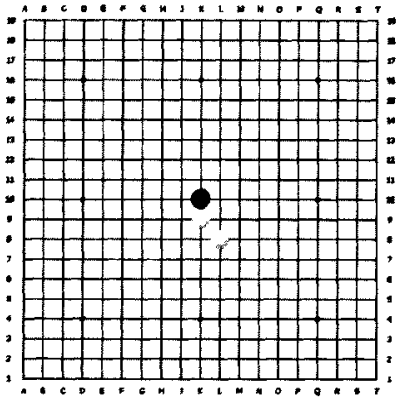

图 2.3 山日局
Figure2.3 shan-ri start

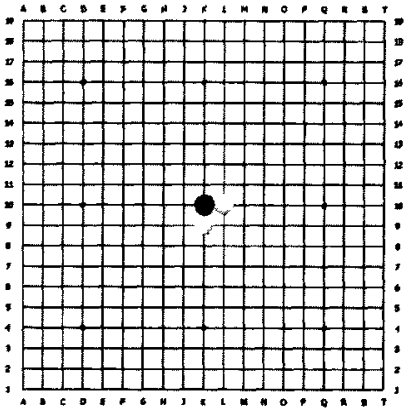

图 2.4 山山局
Figure2.4 shan-shan start

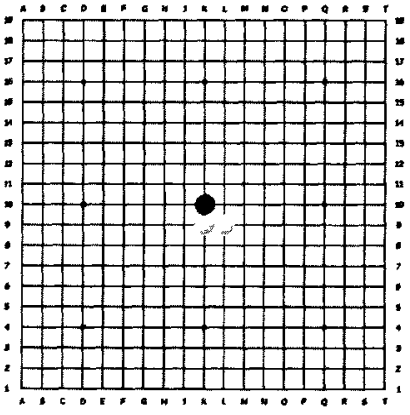

图 2.5 山水局
Figure2.5 shan-shui start

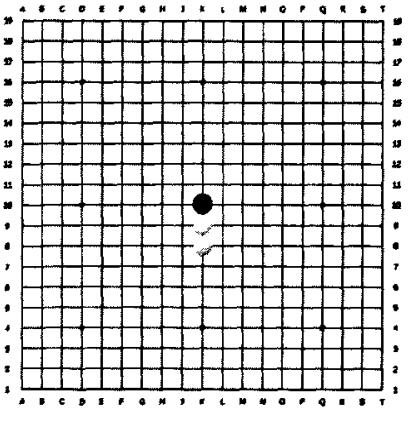

图 2.6 山心局
Figure2.6 shan-xin start

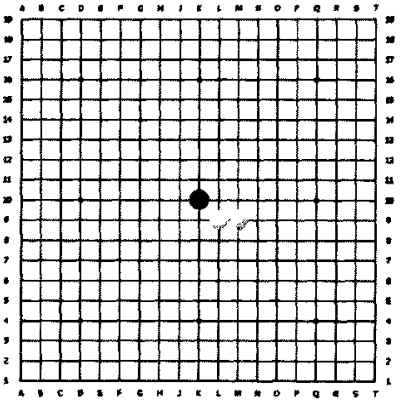

图 2.7 水日局
Figure2.7 shui-ri start

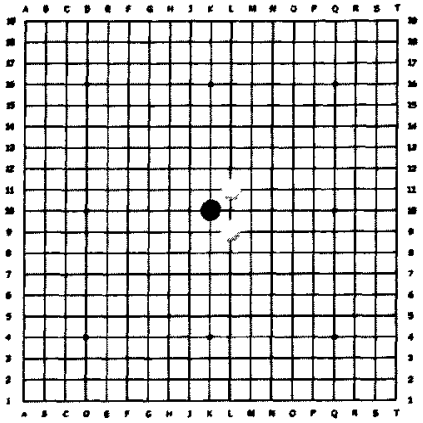

图 2.8 水水局

Figure2.8 shui-shui start

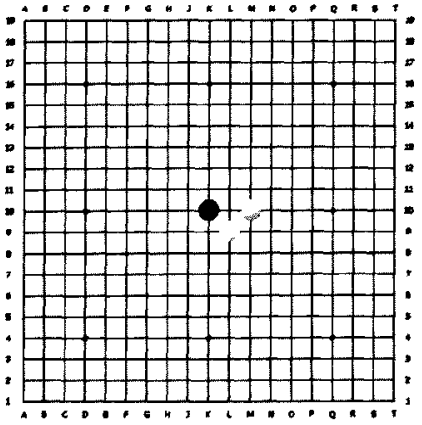

图 2.9 水心局

Figure2.9 shui-xin start

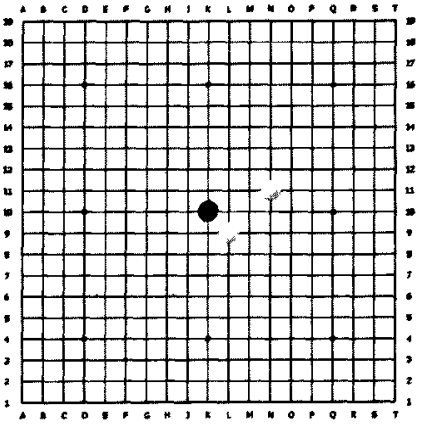

图 2.10 水月局

Figure2.10 shui-yue start

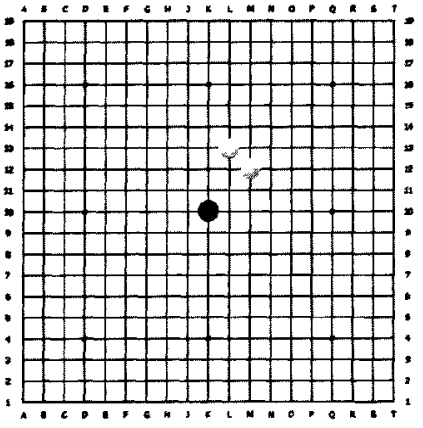

图 2.11 田月局

Figure2.11 tian-yue start

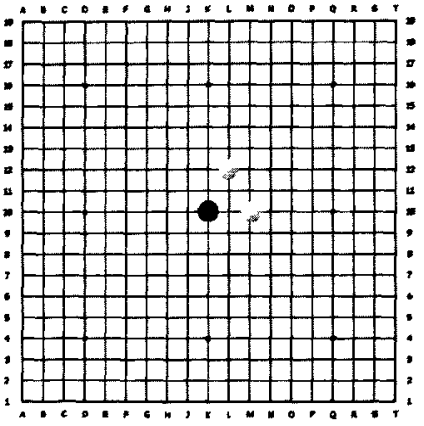

图 2.12 心日局

Figure2.12 xin-ri start

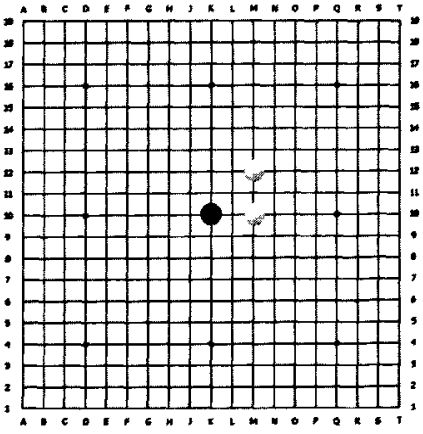

图 2.13 心田局

Figure2.13 xin-tian start

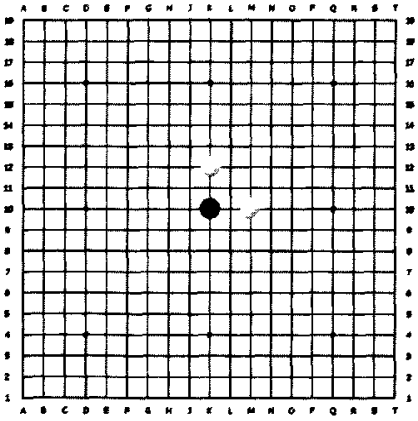

图 2.14 心心局

Figure2.14 xin-xin start

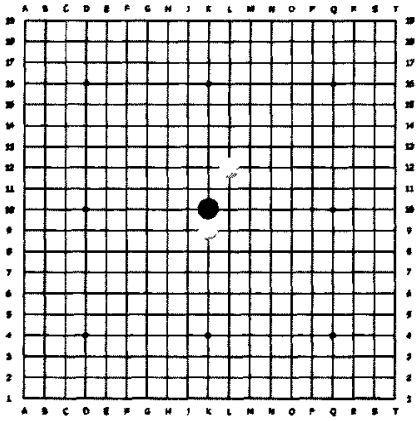

图 2.15 远山日局

Figure2.15 long-shan-ri start

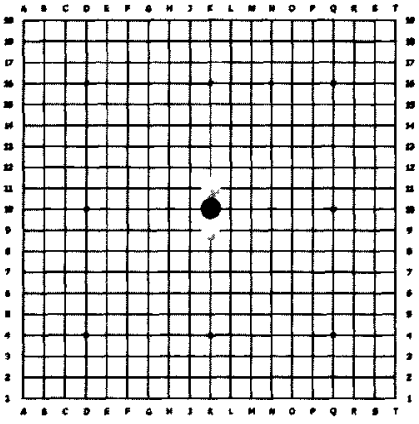

图 2.16 远山山局

Figure2.16 long-shan-shan start

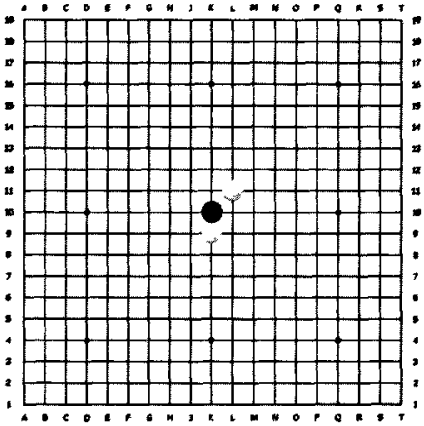

图 2.17 远山水局

Figure2.17 long-shan-shui start

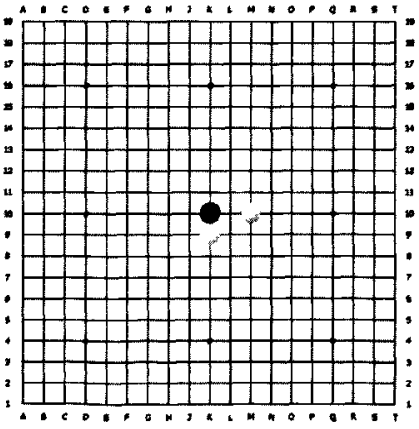

图 2.18 远山心局

Figure2.18 long-shan-xin start

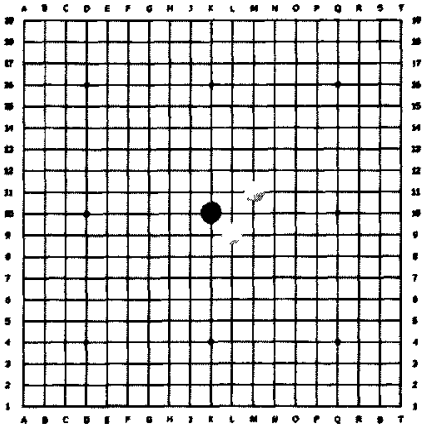

图 2.19 远水日局

Figure2.19 long-shui-ri start

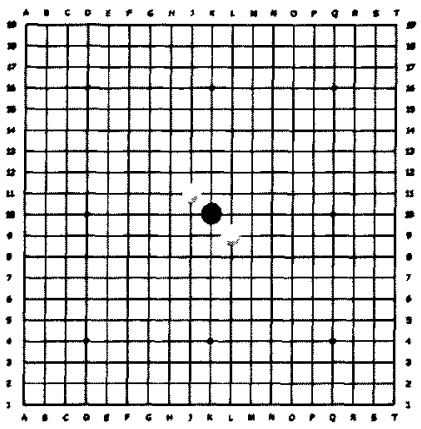

图 2.20 远水水局

Figure2.20 long-shui-shui start

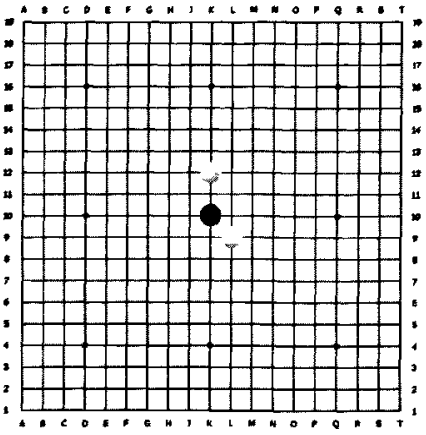

图 2.21 远水心局

Figure2.21 long-shui-xin start

## 2.6 六子棋诘棋

诘棋是指巧妙设计的盘面，解题者必须很有技巧地找出问题所要求达到的目的，例如必胜下法、阻挡下法。围棋、五子棋都称为“诘棋”，象棋多称为“解残局”。当然，六子棋也有，并沿用围棋、五子棋的说法，称之为诘棋。但有些六子棋的诘棋并非只有唯一的解，对于这样的情形，答案尽量选择较为简易的解法。

## 2.7 六子棋发展动向

### 2.7.1 台湾六子棋协会

台湾六子棋协会于 2006 年 12 月正式获得台湾内政部审核筹备，并于 2007 年 4 月 15 日正式成立。以推广六子棋教育、提高棋艺文化水准为宗旨，并明确了相关的任务。

### 2.7.2 第十一届奥林匹亚计算机赛局竞赛中六子棋比赛之结果

参与队伍：EVG: 前台大教授许舜钦之团队。

NCTU6 (交大六号): 六子棋研究小组。

X6: 刘思源及颜士净。

比赛地点：意大利杜林 (Torino, Italy)

比赛时间：2006 年 5 月 26 日

比赛方式：双循环赛

比赛结果：第一名：NCTU6 （六胜二和）

第二名：X6 （二胜二和四负）

第三名：EVG （二胜六负）

## 2.8 本章小结

本章介绍了六子棋及其平台的相关概念，总结了六子棋的特点，关注着六子棋的发展动向，为本文研究六子棋计算机博弈奠定了良好的基础。

# 3 棋类设计的通用方法和思想——计算机博弈

[本章摘要]本章简单介绍了计算机博弈的要点，说明了人机对弈程序组成部分，简单分析了每个部分各自的作用与功能。

## 3.1 计算机博弈的要点

六子棋是"信息完备"的游戏，因为游戏双方面对的局面是同一个局面，任何一方所掌握的棋子及其位置的信息是一样的。除了六子棋，象棋、围棋等也可属于这一范畴。[13]

人机对弈的程序，至少应具备以下 5 个部分：[19]

① 某种在机器中表示棋局的方法，能够让程序知道博弈的状态。

② 产生合法走法的规则，以使博弈公正地进行，并可判断人类对手是否乱走。

③ 从所有合法的走法中选择最佳的走法技术(博弈搜索)。

④ 一种评估局面优劣的方法(评估函数)，用以同上面的技术配合做出智能的选择。

⑤ 一个人机界面，有了他，计算机博弈程序才能使用。

## 3.2 计算机博弈程序的组成

### 3.2.1 人机界面

任何计算机博弈程序都需要一个人机界面，由于六子棋刚刚兴起，还没有类似象棋中的 ucci 之类的统一的界面协议和引擎，现在都还是各自开发各自的。总体来说，人机界面是为了方便棋手的操作，并且提供难度选择、计时器、走棋记录等功能。

### 3.2.2 棋盘和棋局表示－数据结构

要让计算机下棋首先需要解决的就是棋盘和棋局的数字表示。主要有以下几种：

① 比特表示（位棋盘）

常用于在着法生成过程中，时常只关心棋子的分布，而不关心是什么棋子，比如六子棋就属于这类。

位棋盘其实就是一个 N 位长度的变量，用来记录棋盘上的某些布尔值。因为棋盘上有 N 格，所以 N 位正好对应它的 N 格。

另一种比特棋盘位表示方法是棋盘状态矩阵 S 的布尔表示。定义为：

$$B=[b_{i,j}]_{a\times b},\ b_{i,j}=1,\ s_{i,j}\neq 0;\ b_{i,j}=0,\ s_{i,j}=0$$

② 矩阵表示

比如棋盘上面有9路9行形成81个交叉点，它很容易用9×9 的棋盘数组矩阵 $M_{9\times9}$ 表示与坐标的对应关系。

要表示棋局则首先要给棋子编码。应该说编码的方法是任意的，只要能够满足棋局表示的唯一性和可区分性，都是可行的编码。如果考虑和追求棋局处理的节俭与快捷，那么在编码上还是具有研究的余地。

以六子棋为例，黑子编码为0，白子编码为0。

更复杂一些的还有兵种编码和棋子编码，矩阵有兵种状态矩阵和棋子状态矩阵，以"齐天大圣"为例：

矩阵为10×9的矩阵

象棋兵种的中英文表示与棋天大圣的兵种编码表

| 国象兵种 | King | Rook | Knight | Cannon | Queen | Minister | Pawn |
|---|---|---|---|---|---|---|---|
| 中象兵种 | King | Rook | Horse | Cannon | Bishop | Elephant | Pawn |
| 红子 | 帅 | 车 | 马 | 炮 | 仕 | 相 | 兵 |
| 字母代号 | k | r | h | c | b | e | p |
| 棋天大圣编码 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 黑子 | 将 | 车 | 马 | 炮 | 士 | 象 | 卒 |
| 字母代号 | K | R | H | C | B | E | P |
| 棋天大圣编码 | -1 | -2 | -3 | -4 | -5 | -6 | -7 |

图 3.1 象棋兵种编码

Figure3.1 Chess arm coding

棋子(Chessman)编码(编号)表

| $j$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 黑方 | 将 | 黑车 | | 黑马 | | 黑炮 | | 黑士 | | 黑象 | | 黑卒 | | | | |
| $j$ | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 |
| 红方 | 帅 | 红车 | | 红马 | | 红炮 | | 红仕 | | 红相 | | 红兵 | | | | |

图 3.2 象棋棋子编码

Figure3.2 Chinese Chess chessman coding

$$S_0^B=\begin{bmatrix} -2 & -3 & -5 & -6 & -1 & -6 & -5 & -3 & -2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -4 & 0 & 0 & 0 & 0 & 0 & -4 & 0 \\ -7 & 0 & -7 & 0 & -7 & 0 & -7 & 0 & -7 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 7 & 0 & 7 & 0 & 7 & 0 & 7 & 0 & 7 \\ 0 & 4 & 0 & 0 & 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 3 & 5 & 6 & 1 & 6 & 5 & 3 & 2 \end{bmatrix}$$

图 3.3 兵种状态矩阵

Figure3.3 Arm State Matrix

$$S_0^M=\begin{bmatrix} 2 & 4 & 10 & 8 & 1 & 9 & 11 & 5 & 3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 6 & 0 & 0 & 0 & 0 & 0 & 7 & 0 \\ 12 & 0 & 13 & 0 & 14 & 0 & 15 & 0 & 16 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 28 & 0 & 29 & 0 & 30 & 0 & 31 & 0 & 32 \\ 0 & 22 & 0 & 0 & 0 & 0 & 0 & 23 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 18 & 20 & 26 & 24 & 17 & 25 & 27 & 21 & 19 \end{bmatrix}$$

图 3.4 棋子状态矩阵

Figure3.4 Chessman State Matrix

### 3.2.3 着法生成[20]

着法生成方法一般有棋盘扫描法、模板匹配法和预置表法，时常还结合使用。

① 棋盘扫描法

在着法生成的过程中需要在棋盘上反复扫描有效区域、制约条件和落子状况，确定有那些地址可以落子。

根据六子棋的规则，棋盘有效区域内的所有空白的交点都是可行落子点。六子棋、五子棋、围棋一类的棋类设计中最常用。（只是对一般规则，比如职业五子棋对弈有三·三、四·四等禁手的规则）

在着法的表达上，棋盘扫描法最为直观，但时间开销巨大。

② 模板匹配法

以象棋为例，当动子确定之后，其落址与提址的相对关系便被固定下来。于是可以为某些动子设计“模板”，只要匹配到提址，便可以迅速找到落址。下图给出了走马的匹配模板。当马对准提址，×表示蹩马腿的制约条件，○表示符合“走日”规则的落址，根据×的具体分布，很容易判断可能的落址。再通过单项比特矩阵比对，实现“本方子则止，对方子则吃”，完成“提 2 动 2 落 2 吃”内容的确定。

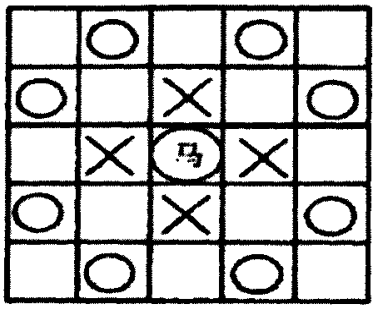


图 3.5 走马匹配模板

Figure3.5 Horse matching template

然而在六子棋中，不存在提址问题，只有落子位置，而且所有当前无子（空）的位置都是有效的走法，因此在六子棋中用模版匹配法并不实用。

③ 预置表法

预置表法是使用最为经常的着法生成方法。它的基本思想就是用空间换时间。为了节省博弈过程中的生成着法的扫描时间，将动子在棋盘任何位置(提址)、针对棋子的全部可能分布，事先给出可能的吃子走法和非吃子走法。当然，六子棋无吃子情况。

### 3.2.4 机器博弈、搜索技术

① 博弈树[21]

任何棋类游戏都要定义一棵有根的树(即“博弈树”)，一个结点就代表棋类的一个局面，子结点就是这个局面走一步可以到达的一个局面。例如下图是井字棋(Tic-tac-toe)的博弈树：

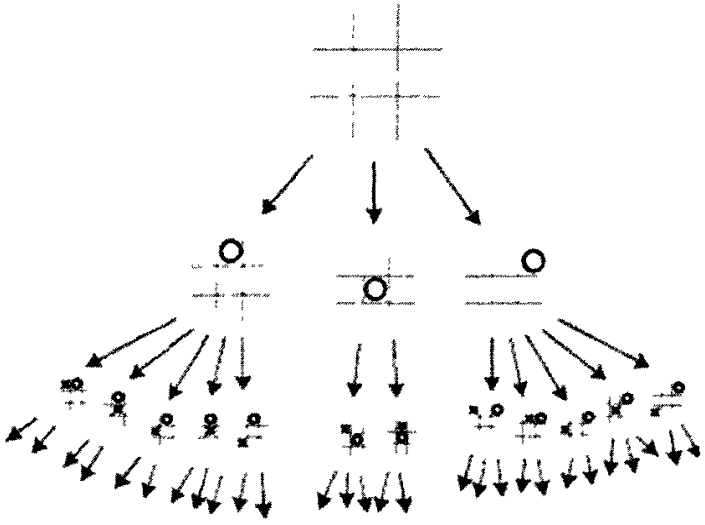

图 3.6 井字棋博弈树

Figure3.6 Tic-tac-toe game-tree

实际上，这个博弈树的根结点应该有 9 个子结点，但是去掉了一些对称的情况。如果同样的棋盘是由两个不同的着法顺序形成的，那么我们就建立两个结点，所以这的确是树的结构。另外我们假设棋手是轮流下棋的，没有人一次走多步或跳过不走的，那些复杂的情况（比如六子棋）可以把它走的一系列着法看作一个着法来处理。最后，我们假设博弈树是有限的，这样我们就不会遇到永无止境的棋局或者一步有无限多种着法的棋局。

一般搜索树中有三种类型的结点：

1） 偶数层的中间结点，代表棋手甲要走的局面；

2） 奇数层的中间结点，代表棋手乙要走的局面；

3） 叶子结点，代表棋局结束的局面，即棋手甲或棋手乙获胜，或者是和局。

② 搜索算法

搜索算法是博弈树求解的灵魂，只有有了有效的搜索算法才能在有限的时间内找到正确的解。搜索法是求解此类图模型的基本方法。我们无法搜索到最终的胜负状态，于是搜索的目标便成为如何在有限深度的博弈树中找到评估值最高而又不剧烈波动的最佳状态(棋局)，于是从当前状态出发到达最佳状态的路径便为最佳路径(Principal continuation)，它代表着理智双方精彩对弈的系列着法。而最佳路径上的第一步棋便成为当前局面的最佳着法(The best move)。所谓“不剧烈波动”就是说最佳棋局不是在进行子力交换与激烈拼杀的过程当中。需要注意的是博弈树不同于一般的搜索树,它是由对弈双方共同产生的一种“变性”

搜索树。

1） 极大极小搜索[22] [23] [24] [25]

香侬(Claude Shannon) 教授早在1950年首先提出了“极大-极小算法”(Minimax Algorithm)，从而奠定了计算机博弈的理论基础。

我们考虑两个玩家对弈，分别为MAX 和MIN。MAX先走，之后两人交替走步直到游戏结束。游戏用产生式系统描述。由于不可能对完整解图进行搜索，我们定义一个静态估计函数f，以便对游戏状态的当前势态进行估值。一般规定有利于MAX 的状态取f(s)>0，有利于MIN的状态取f(s)<0。用井字棋为例给出极大极小搜索的5个步骤。

a. 生成整个博弈树，即扩展树的每个节点。

b. 用静态估值函数f 对每个叶节点进行估值，得出每个终节点的评价值。

c. 用终节点的估值来得到其搜索树上一层节点的估值。如图3.7所示A1走步的MIN结点选择走步，应选A15 ,由A15 走步得到的估值是A1走步中最小的，同样A2 、A3 的MIN 也选择最小值走步。

d. 重复(3) 过程在MAX 层取其分支的最大值，MIN 层取其分支的最小值，一直回溯到根结点。

e. 最终，根结点选择分支值最大的走步走，如下图中，A3 走步为根的最佳走步。

这就是极大极小搜索过程(MINIMAX decision)。

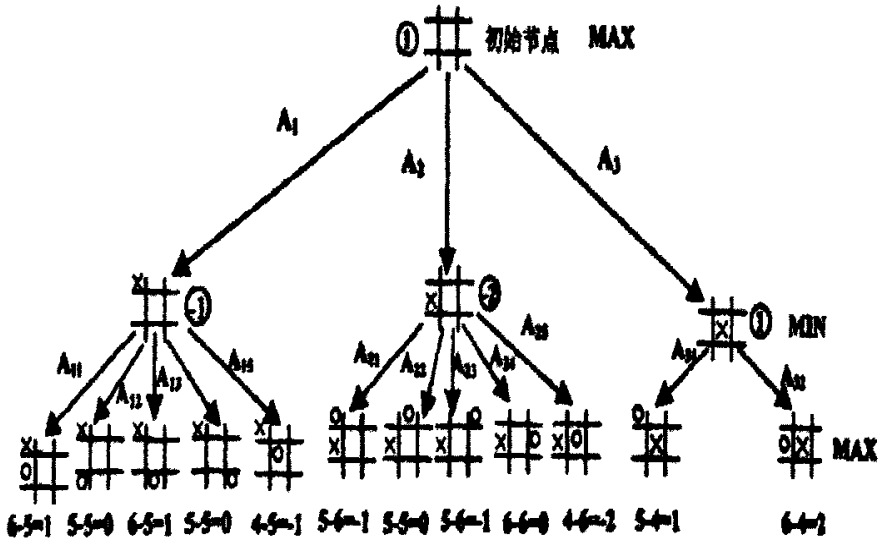


图 3.7 井字棋的极大极小搜索过程

Figure3.7 Tic-tac-toe MINIMAX decision

井字棋的一阶搜索树，深度为 3，其中估计函数 f 的值为棋盘上剩余格全用 MAX 填满所成的三子一线的个数 m 减去棋盘上剩余格全用 MIN 填满所成的三子一线的个数 n ,即 f(s)= m – n。

2) α-β 剪枝搜索[26] [27] [28] [29] [30] [31] [32] [33]

剪枝：我们假设五子棋用一个静态估值函数，用 MINIMAX 过程对博弈树进行搜索，每一次扩展 15×15=225 个结点，假设一个性能好的程序在一台普通的 PC 机上，一秒内可搜索 10 000 个结点，对于三层的 MINIMAX 搜索树需扩展 11 390 625 个结点，则需 1 139s 约 19min 的时间下一步棋。这显然是不可忍受的。幸运的是，我们不用把每个结点都搜索一遍也可获得和 MINIMAX 搜索同样结果的走步。不搜索分支结点而舍弃该分支的过程称为剪枝。

α-β 剪枝搜索：α-β 剪枝搜索是一种基于 α-β 剪枝(α-β cut-off)的深度优先搜索(Depth-first search)。它去掉了一些不影响最终结果的分支而返回与 MINIMAX 相同走步的过程。

我们不妨将走棋方定为 MAX 方，因为它选择着法时总是对其子节点的评估值取极大值，即选择对自己最为有利的着法；而将应对方定为 MIN 方，因为它走棋时需要对其子节点的评估值取极小值，即选择对走棋方最为不利的、最有钳制作用的着法。

α 剪枝：在对博弈树采取深度优先的搜索策略时，从左路分枝的叶节点倒推得到某一层 MAX 节点的值，可表示到此为止得以"落实"的着法最佳值，记为 α。显然此 α 值可作为 MAX 方着法指标的下界。在搜索此 MAX 节点的其它子节点， 即探讨另一着法时，如果发现一个回合(2 步棋)之后评估值变差，即孙节点评估值低于下界 α 值，则便可以剪掉此枝(以该子节点为根的子树)，即不再考虑此"软着"的延伸。此类剪枝称为 α 剪枝。下图给出了搜索和剪枝过程，最后得到如粗箭头所示的最佳路径片断。

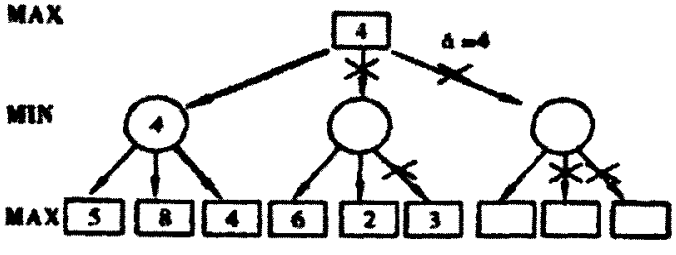


图 3.8 α 剪枝

Figure3.8 αPruning

β剪枝：同理，由左路分枝的叶节点倒推得到某一层 MIN 节点的值，可表示到此为止对方着法的钳制值，记为 β。显然此 B 值可作为 MAX 方可能实现着法指标的上界。在搜索该 MIN 节点的其它子节点，即探讨另外着法时，如果发现一个回合之后钳制局面减弱，即孙节点评估值高于上界 β 值，则便可以剪掉此枝，即不再考虑此"软着"的延伸。此类剪枝称为 β 剪枝。下图给出了搜索和剪枝过程，最后得到如粗箭头所示的最佳路径片断。

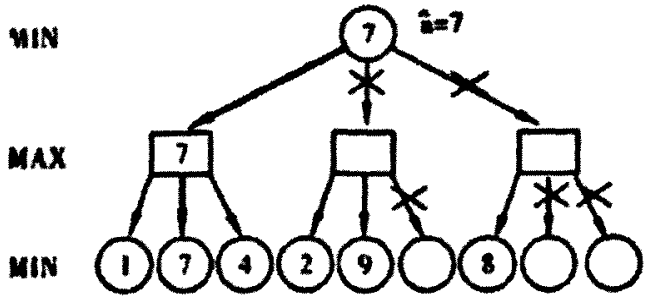


图 3.9 β 剪枝
Figure3.9 βPruning

需要指出的是，α-β剪枝是根据极大极小搜索规则的进行的，虽然它没有遍历某些子树的大量节点，但它仍不失为穷尽搜索的本性。剪枝技巧的发现，一下便为博弈树搜索效率的提高开创了崭新的局面。

1975 年Knuth和Moore 给出了α-β算法正确性的数学证明。α-β剪枝算法的效率与子节点扩展的先后顺序相关。

为了得到最好的节点扩展顺序，许多搜索算法在着法(节点扩展的分枝) 排序上给予特别的关注。比如在着法生成(节点扩展) 时，先生成吃子着法，尤其先生成吃分值高的"大子"着法，因为由此产生着法更有可能是最佳的.围绕着法排序，已经出现许多优秀的搜索算法与举措。如：同形表法(Transposition table)、吃子走法的SEE 排序、杀手走法(Killer heuristic)、未吃子走法的历史启发排序(Historic heuristic)、类比法(Method of analogies)等。

有人将α-β剪枝作用延伸到多个回合之后，于是又出现了深层α-β剪枝(Deep α-βcut-off) 算法，也取得很好效果。

3) α-β 窗口搜索[34]

从 α-β 剪枝原理中得知，α 值可作为 MAX 方可实现着法指标的下界，而 β 值(应对方的钳制值) 便成为 MAX 方可实现着法指标的上界，于是由 α 和 β 可以形成一个 MAX 方候选着法的窗口。围绕如何能够快速得到一个尽可能小而又尽可能准确的窗口，也便出现了各种各样的 α-β 窗口搜索算法。如 Fail-Soft

Alpha-Beta、Aspiration Search (渴望搜索)、Minimal Window Search (最小窗口搜索)、Principal Variable Search (PVS 搜索)/ Negescout 搜索、宽容搜寻(Tolerance Search)等。

4) 迭代深化搜索[35]

不难想象，深度为 D21 层的最佳路径，最有可能成为深度为 D 层博弈树的最佳路径。Knuth 和 Moore 分析表明，对于分枝因子为 B 博弈树，利用 α-β 剪枝搜索 D 层所需时间大约是搜索 D-1 层所需时间的√B 倍。如果国象取 B=36，每多搜索一层就要花上原先的 6 倍时间。于是 CHESS4.6 和 DU CHEN SS 课题组开始采用逐层加深搜索算法。先花 1/6 的时间做 D-1 层的搜索，找到最佳路径，同时记载同形表、历史启发表、杀手表等有价值信息，以求达到 D 层最好的剪枝效果，可谓"磨刀不误砍柴功"。

目前几乎所有高水平的博弈程序都采用迭代深化算法，并在不断改进。如 PV 记录(Principal Variation), 以及和渴望窗口搜索(Aspiration Windows Search)的结合，都会对走法排序产生非常好的效果。另外，逐层加深的搜索算法比固定深度搜索算法更适合于对弈过程的时间控制。

5) 启发式搜索(Heuristic search) [36]

具体问题的领域决定了初始状态、算符和目标状态，进而决定了搜索空间。因此，具体问题领域的信息常常可以用来简化搜索。此种信息叫做启发信息，而把利用启发信息的搜索方法叫做启发性搜索方法。

6) 负极大值算法[31] [32]

前面谈到博弈树的搜索是一种"变性"搜索。在偶数层进行"Max 搜索"，而在奇数层进行"Min 搜索"。这无疑给算法的实现带来一大堆麻烦。

Knuth 和 Moore 充分利用了"变性"搜索的内在规律，在 1975 年提出了意义重大的负极大值算法。它的思想是：父节点的值是各子节点值的变号极大值，从而避免奇数层取极小而偶数层取极大的尴尬局面。

$$F(v) = \max\{-F(v\,1), -F(v\,2)\ldots, -F(v\,n)\}$$

其中，$v\,1, v\,2, \ldots, v\,n$ 为 v 的子节点。

从以上有限的介绍不难看出，博弈树的搜索算法丰富多彩，改革、重组与创新的余地很大，一定会成为机器博弈研究的重点。

### 3.2.5 评估函数[19] [37]

对于博弈树求解有了良好的搜索算法还只是问题的一个方面，问题的另一个方面就是评估函数。只有有了良好的评估函数才能保证较快地找到正解。而评估函数是对棋局的综合评估，该函数的好坏直接决定解题能力强与弱。通常一个优秀的棋手总有一个良好的对棋局的判断能力，能够协调各棋子的关系、

取舍，有机地组织各棋子的进攻步调，控制棋局的发展。因此如果要把这一整套的思维物化成一个数值函数来评估，本身就是一个相当复杂的问题。

① 整体考虑

评估函数综合了大量跟具体棋类有关的知识。我们从以下两个基本假设开始：

1) 我们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做出的估计。

2) 我们衡量的这个性质应该跟对手衡量的性质是一样的，如果我们认为我们处于优势，那么反过来对手认为他处于劣势。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。

评估可以是简单的或复杂的，这取决于在程序中加了多少知识。评估越复杂，包含知识的代码就越多，程序就越慢。通常，程序的质量(棋力)可以通过知识和速度的乘积来估计，如下图所示：

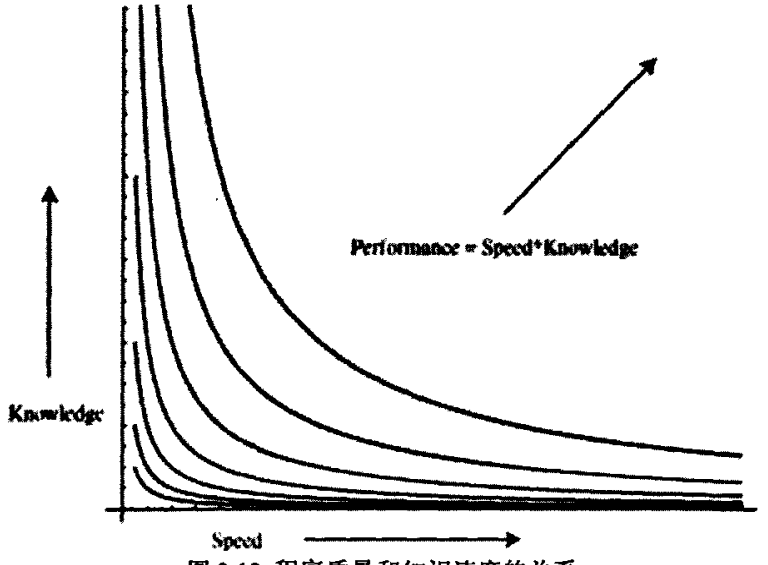


图 3.10 程序质量和知识速度的关系

Figure3.10 Relationship of the performance with knowledge and speed

② 组合评估要素

把评估要素组合起来，评估函数是很多项的和，每一项是一个函数，它负责找到局面中的某个特定因素。

棋类程序应该充分尝试各种可能的评估函数：把各种胜利的可能性结合起来，包括很快获胜(考虑进攻手段)，很多回合以后能获胜，以及在残局中获胜(国际象棋中就必须考虑通路兵的优势)的可能性，然后把这些可能性以适当的方式结合起来。如果黑方很快获胜的可能性用 bs 表示，而白方用 ws，在很多回合以

后获胜(即不是很快获胜)的可能性是 bm 或 wm，而在残局中获胜的可能性是 be 或 we，那么整个获胜的可能性就是：

黑方：bs + (1 - bs - ws) * bm + (1 - bs - ws - bm - wm) * be

白方：ws + (1 - bs - ws) * wm + (1 - bs - ws - bm - wm) * we

通过和类似上面的公式把若干单独概率结合起来，在评估函数中或许是个很好的估计概率的思路。每种概率是否估计得好，这就需要用程序的估计来和数据库中棋局的真实结果来做比较，这就需要让程序具有基本判断的能力(判断某种攻击是否能起到效果)。

③ 评估函数中所需加入的信息

典型的评估函数，要把下列不同类型的知识整理成代码，并组合起来：

1) 子力(Material)：在国际象棋中，它是子力价值的和，在围棋或黑白棋中，它是双方棋盘上棋子的数量。这种评价通常是有效的，但其他像五子棋一样的游戏，子力是没有作用的，因为好坏仅仅取决于棋子在棋盘上的位置，看它是否能发挥作用。

2) 空间(Space)：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。

3) 机动(Mobility)：每个棋手有多少不同的着法？有一个思想，即你有越多可以选择的着法，越有可能至少有一个着法能取得好的局势。这个思想在黑白棋中非常有效，国际象棋中并不那么有用。

4) 着法(Tempo)：这和机动性有着密切的联系，它指的是在黑白棋或连四子棋中(以及某些国际象棋残局中)，某方被迫作出使局面变得不利的着法。和机动性不同的是，起决定作用的是着法数的奇偶而不是数量。

5) 威胁(Threat)。对手是否会有很恶劣的手段？你有什么很好的着法？例如在国际象棋或围棋中，有什么子可能要被吃掉？在五子棋或连四子棋中，某一方是否有可以连起来的子？在国际象棋或西洋棋中，有没有子将会变后或变王？在黑白棋中，一方是否要占角？这个因素必须根据威胁的远近和强度来考虑。

6) 形状(Shape)。在围棋中，如果连起来的一串子围成两个独立的区域(称为“眼”)，那么它们就是安全的。在国际象棋中，并排的兵通常要比同一列的叠

兵强大。形状因素是非常重要，因为局面的长远价值在几步内不会改变，也不会因为搜索而变化，这正是形状因素需要衡量的。(搜索可以找到短期的手段来改进局面，所以评价本身需要包括更多的长远眼光，使得搜索可以察觉到。)

7) 图案(Motif)。一些常见的具有鲜明特点的图案，蕴涵着特殊的意义。例如在国际象棋中，象往往可以吃掉边兵，却会被边上的兵困住。当象被困住时，对手可能还需要很多步才会吃掉它，因此被困的情形不容易被计算机的搜索程序所发现。有些程序通过特殊的评价因素来警告电脑，吃掉那个边兵可能会犯错误。在黑白棋中，在角落的邻近格子上放一个子来牺牲一个角，往往是非常有用的，这样当对手占领这个角时，就可以在这个子的边上放一个提不掉的子，从而在另一个角上取得优势。

④ 获取评估函数中数值的方法

局面评价中的很多函数，把这些函数加起来就可以组合成评估函数。但是数值从哪里来呢？以下给出一些获取评估函数中数值的方法：

1) 手工方法

a. 规格化(Normalize)。如果你只关心评价的顺序，而通常不怎么关心评价值，那么你就可以把每一项都乘以同样的常数。这就意味着你对某个特定的项目(比如说兵的价值)可以硬性设一个值，其他值就表示成它们相当于多少个兵。这个做法可以让你减少一个需要设定的参数。

b. 约束法(Deduce Constraints)。你希望让电脑做出什么样的判断，考虑这些问题就可以确定一些参数了。例如在国际象棋中，即使你赚到一个兵，用车换象或马通常还是坏的，但是如果你赚到两个兵那还是好的，因此子力价值要满足 R>B+P(防止换单兵)和 R<B+2P(鼓励换双兵)。这样的不等式你给得越多，合适的权重组合就越少。在一开始设定权重值的时候，这个方法通常可以得到合适的值，但是后面你仍然需要做一些调整。

c. 交手法(Hand Tweaking)。这是很常用的方法，仅仅是让你的程序对弈足够多的次数，来找到它的优势和弱点，猜测哪些参数会让程序更好，然后挑选新的参数。这个方法可以很快得到合理的结果，但是你需要对这种棋类有足够的了解，这样就可以根据程序的对局来做分析，知道程序的问题在哪里。

2) 不需要人工干预的方法

a. 爬山法(Hill-Climbing)。类似于交手法，每次对权重作很小的改变，测试改变后的表现，仅当成绩提高时才采纳这个改变，需要重复很多次。这个方法看上去很慢，并且只能找到"局部最优"的组合(即评价可能很差，但是任何很小的改变都会使评价更差)。

b. 模拟退火法(Simulated Annealing)。类似于爬山法，也是对权重做出改变来提高成绩的。但是如果改变没有提高成绩，有时候(随机地，给定一个几率)也采纳改变，试图跳出全局最优。这个方法需要给定一些几率，从几率高、梯度大的条件开始，然后逐渐减小。模拟退火法比爬山法更慢，但是最终可能得到比较好的值。

c. 遗传算法(Genetic Algorithms)。爬山法和模拟退火法可以得到一组好的权重，它们是逐渐变化的。相反，遗传算法可以得到几组不同的好的权重，不断增加新的组合跟原来的做比较(取用某组中的某个权重，另一组中的另一个权重，互相交换得到新的)，通过淘汰坏的组合来控制种群的数量。

d. 神经网络(Neural Networks)。实际上这更多地是一种评价函数的类型，而不是用来选择权重的：神经元是阈值(输入权重的和)的函数，第一层神经元输入的关于局面的性质(例如位棋盘表示中的某几个位)就可以构造网络，然后前一层的结果输入到后一层。因此单输入神经元的单层网络就等同于我们上次讨论过的一阶评价函数，但是接下来就可以构造更复杂的神经网络了，而且用这种方法作为评价函数是不难的(只要根据输入的改变来重新计算神经元的输出就可以了)。问题仍然像前面所说的，如何设置权重？除了前面的方法外，针对神经网络还发展出一些方法，例如“暂时差别学习”(Temporal Difference Learning)。其基本思想是确定网络何时会作出坏的评价，并且让每个权重增加或减小看是否会评价得更好，这很类似于爬山法。跟其他自动学习的方法相比，神经网络的好处就在于它不需要很多人类的智慧：你不需要懂得太多的棋类知识，就可以让程序有个比较好的评价函数。但是根据目前我们掌握的情况，根据自己的智慧来做评价函数，要比机器学习做得好，并且做得快。

## 3.3 本章小结

本章总结了现有棋类游戏计算机博弈的要点和方法，说明了计算机博弈程序组成部分的作用与功能，为后文六子棋计算机博弈极其系统的研究提供了思路。

# 4 六子棋计算机博弈系统的平台构建

[本章摘要] 本章将综合前两章的内容，提出了六子棋计算机博弈系统平台构建的方法，内容主要包括棋盘、棋局状态等数据结构的表示方法，搜索引擎模块所采用的搜索算法以及走法生成模块中，对搜索的结果进行分析和处理，提出了一种"两步"到"一步"的处理方法，解决了一个综合搜索的问题。

## 4.1 引言

棋盘以及棋局状态等棋盘状态空间的计算机表示方法基本上都是用二维数组和结构等数据结构来表示。

目前，搜索算法以及走法生成都是一个相对比较成熟的技术。各种搜索算法和走法生成技术都广泛应用于不同棋类的计算机博弈研究中。但是单纯依靠一种搜索算法来实现某种棋类的计算机博弈，效果是不尽如人意的。

本文主要是在六子棋计算机博弈系统搜索引擎模块，以 α-β 剪枝搜索算法为基础，同时在里面加入了一些启发式信息，以提高搜索的效率和准确性。

在走法生成模块，针对于六子棋的特殊性，解决了"两步"到"一步"的综合搜索并根据搜索结果确定落子的问题。

## 4.2 棋盘状态空间表示

要让计算机学会下棋，首先就要把下棋问题表示成计算机可理解的形式，即

把六子棋问题形式化，存在计算机中，并能让搜索、估值等算法对这些数据进行操作。需要在计算机中表示的主要问题有棋盘局势状态、落子的顺序表示等。

### 4.2.1 棋盘局势状态表示

棋盘表示主要探讨的是使用什么样的数据结构来表示棋盘上的信息。一般来说，这与具体的棋类知识密切相关。通常，用来描述棋盘及其上棋子信息的是一个二维数组。要使计算机知道棋盘局势状态，就是要它记住棋盘中哪些位置有黑棋，哪些位置有白棋，以及哪些位置还未走棋。

本系统首先定义一个类，表示棋盘中某个位置的信息。

```
public class StepStatus
    {
        public bool used;      //是否已走棋
        public int player;     //是黑棋还是白棋
```

```
        public StepStatus()
        {
        }
        public StepStatus(StepStatus stepStatus)
        {
            this.used = stepStatus.used;
            this.player = stepStatus.player;
        }
    }
```

然后声明此类的对象数组：

```
StepStatus[][] chessBoardStatus = new StepStatus[19][19];
```

这样就可以表示整个棋盘的状态信息。

接着在系统中定义了一个结构用来表示一步棋的棋子位置以及是何种棋：

```
public struct Step
    {
        public int x;          //横坐标
        public int y;          //纵坐标
        public int player;     //是白棋还是黑棋
    }
```

### 4.2.2 落子的顺序表示

棋盘中的落子有先后顺序，六子棋是黑棋先下一颗子后，从白棋开始以后双方各走两颗子，直到决出胜负或棋盘走满为止。那么这个下棋顺序也需要表示出来，本系统用堆栈结构来存储棋子及其坐标，先下的棋子放在栈低端，后下的放在高端，通过对栈的访问，可以得到下棋的顺序。

例如，下图所示的栈结构表示这样的 4 颗棋分别下在棋盘上：

| | | | |
|---|---|---|---|
| 12 | 11 | 1 | ←栈顶 |
| 11 | 11 | 1 | |
| 11 | 10 | 0 | |
| 11 | 9 | 0 | ←栈底 |

图 4.1 栈结构

Figure4.1 stack

1- 黑棋下在（11，9）即（K，9）位置

2- 黑棋下在（11，10）即（K，10）位置
3- 白棋下在（11，11）即（K，11）位置
4- 白棋下在（12，11）即（L，11）位置

在使用了这样的栈结构表示落子的顺序后，双方的下棋就可以用入栈操作来完成，而有时候需要悔棋，返回上一步的状态，那么就可以用出栈操作来完成。同时这样的结构可以记录棋局，方便以后复盘学习使用。

## 4.3 六子棋计算机博弈问题描述

几乎所有的棋类问题，都可以用博弈树来描述。博弈树是把计算机和用户所有可能的走法和局面罗列出来的一颗树。黑白双方交替地按合理走法把树展开，树的每一个节点都表示某一个特定局面。根节点表示的是当前需要计算的局面，中间节点表示的是对弈过程中的某一个局面，叶子节点是树的最底端，表示可以推导的局面。叶子节点和根节点之间的最大距离,称为搜索深度。整个博弈树描述的是从当前局面出发，包含所有可能的对弈过程的搜索树。六子棋计算机博弈问题也就转化为寻求最佳路径的问题。[38]

对于树中的每一个节点来说，黑白双方都会从子节点中选择最有利于自己的分枝。因为博弈树中值的传递是由下至上的，这就要求对叶子节点表示的局面必须有一个极为准确的打分。对于局面最为准确的估计莫过于已经分出胜负的情况，即建立在叶子节点分出胜负的完全博弈树。六子棋的完全博弈树大概有$10^{765}$个节点，建立这个博弈树已经远远超出了当代计算机的处理能力。惟一的解决方法就是让博弈树扩展到计算机运算可以接受的深度，然后对没有分出胜负的叶子节点给出一个最为准确的打分，表示此局面下取得胜利的可能性。而对于节点的打分就是由评估函数计算得到的。

走法生成模块的功能就是按照六子棋的走法规则生成合理走法将博弈树展开；搜索引擎模块的功能则是尽可能缩小树的规模，避免一切冗余的计算。

## 4.4 搜索引擎

计算机要选择有利于它的最佳下法，就要判断哪种形势对它最为有利。但往往对一个形势的判断是很难做到准确的，特别是一盘棋刚开始的时候，棋盘的形势很不明朗，即使是专家也很难做出准确的判断。为了判断哪种下法最有利，我们通常需要向后多算几步，估计一下多走几步后局面的形势如何。这样的思想被称为“多算性”，也就是说，谁看得越深越远，谁就容易获胜。这种思维方式很自然地被应用在了计算机上。上文已经提到，棋局的不断向后计算，形成了一棵博弈树，“多算性”的思想即是对博弈树的搜索过程，这就是博弈树

的极大极小搜索算法。然后在极大极小值的搜索过程中，遍历了整棵博弈树，每个节点都访问了一次，这样的搜索算法粗糙，搜索量非常大，而使搜索效率低下。

为了尽可能缩小树的规模，避免一切冗余的计算，从而提高搜索效率，必须对博弈树中的节点进行筛选，过滤掉一些不必要搜索的节点。由此，本系统的搜索引擎模块中采用了 α-β 剪枝搜索算法，并且根据六子棋的特点，加入了一些启发式信息。

### 4.4.1 α-β 剪枝搜索算法

本文在第三章中已经对 α-β 剪枝搜索算法的原理及搜索过程作了简单的介绍，现将 α-β 剪枝搜索算法引入到六子棋计算机博弈系统的搜索引擎模块。

首先定义一个 α、β 节点的枚举类型：

```
public enum ALPHABETANODETYPE      //alphabeta 节点类型
    {
        ALPHA，BETA
    }
```

表示搜索中是 α 节点还是 β 节点。

然后定义一个结构用来表示用 α-β 剪枝搜索算法搜索某一方该走哪两颗棋的搜索结果：

```
public struct PossibleStep
    {
        public Step[] step;      //走两步棋
        public int priority;     //该走法的优先级
    }
```

其中，public Step[] step;定义了结构 struct Step 的一个结构数组。

最后定义了一个 α-β 剪枝的递归函数来实现 α-β 剪枝过程：

```
public static PossibleStep AlphaBetaRecursion(StepStatus[][] chessBoardStatus, int
player, int width, int depth,ALPHABETANODETYPE nodeType,int[] depthValues)
//alphabeta 剪枝递归函数
```

其中，α-β 剪枝的递归函数包括 6 个参数：

```
StepStatus[][] chessBoardStatus      //当前棋盘状态
int player                           //是黑方还是白方走棋
int width                            //搜索的宽度
int depth                            //搜索的深度
ALPHABETANODETYPE nodeType           //是 α 节点还是 β 节点
```

int[] depthValues　　//两步棋的打分数组

函数返回值为一个 PossibleStep 的结构，指明某一方该走哪两颗棋。

这里需要指出的是，以上所使用的 α-β 剪枝搜索算法，在博弈树的搜索过程中，假设所有节点的打分或者说估值都是准确的，并且都以该估值为依据。但实际上，这样的估值肯定是有误差的，怎样让估值尽量准确，尽量接近真实值，这是下一章评估函数需要解决的问题。

在上述的 α-β 剪枝搜索算法中，搜索的深度和宽度决定了搜索的时间以及搜索的精度。如果搜索的深度越深，宽度越宽，那么搜索的时间就越长，但是搜索的精度就越高，AI 的智能就越高，但这是以牺牲时间作为代价的；反之，如果搜索的深度越浅，宽度越窄，那么搜索的时间就越短，但是搜索的精度就越低，AI 的智能也相对较低。

由此，需要在搜索时间以及搜索精度上作综合考虑，既不能让搜索的时间过长，也不能让搜索的精度过低。本文在搜索算法的设计中，在保证搜索时间符合要求的情况下，尽可能增加搜索深度和宽度，以提高搜索精度，提高 AI 的智能。程序在设计过程中，也是以搜索的深度和宽度的不同来设定 AI 的智能等级的。

### 4.4.2 启发式信息

在六子棋计算机博弈系统的搜索引擎模块中，已经引入了 α-β 剪枝搜索算法。但综观现有棋类的计算机博弈中，完全靠单一的搜索算法来搜索很难得到理想的搜索结果，达不到理想的效果，因而一般都采用两种或几种搜索算法的相结合的方法，或者以一种搜索算法为主，同时根据棋类知识加入一些启发式信息，以获得较为理想的搜索结果。

本文在以 α-β 剪枝搜索算法为主要搜索算法的基础上，根据六子棋的特点，加入了一些启发式信息，主要是对一些诘棋的判断。如果 α-β 剪枝搜索算法在搜索的过程中，发现当前的局面与预先定义的诘棋的某一个局面相符，那么直接调用其对应的最简单的解法。本系统使用表结构来存储一些典型的诘棋局面和其对应的解法。

在 α-β 剪枝搜索算法通过“剪枝”缩小了博弈树的规模后，同时加入一些基于诘棋的启发式信息可以进一步缩小博弈树的规模，避免一切冗余的计算，使搜索效率和准确性得到了进一步的提高。

## 4.5 走法生成

走法生成是指将一个局面的所有可能的走法罗列出来的那一部分程序，也就是用来告诉其它部分下一步可以往哪里走的模块。各种棋类的规则不同，走

法生成的复杂程度也有很大的区别。一般说来，在一种棋类游戏中，双方棋子的种类越多，各种棋子走法的规则也越多，则在程序中，走法生成的实现就越复杂。

以中国象棋为例，比如象只可以走"田"子，就检查与这个象相关联的相位上是否有自己的棋子，并且要检查其间的相眼上是否有棋子；而兵则要注意是不可后退并在没有过河的情况下只能前进一步，过河后则可以左右走一步。现在假定由一个轮到红方走棋的局面，要列举出红方所有符合规则的走法。首先要扫描棋盘，如果某一位置上是一红方棋子，则根据该棋子的类型找出该棋子的所有可走位置。例如该棋子是马，检查马的纵横和横纵方向上的距离分别为 1 和 2 的所有位置是否有红方棋子（检查所走位置），如某位置没有，则还要检查该方向上与马的横纵距离均为 1 的位置是否有棋子（检查是否蹩马脚）。如没有，前一步所检查的位置就是上述红马的一个合法走步。同样其它棋子也要经历各自的判断程序最后找出所有合法的走步。

在六子棋的对弈程序中，由于双方只有黑白各一种棋子，并且在走子的过程中，没有吃子、提子等规则的存在，双方轮流走子，只要有一方首先在棋盘的水平、垂直、斜线方向上形成连续的六颗棋子，就获得胜利。因此，对于六子棋的走法生成来说，棋盘上的任意空白位置都是合法的走法。但在实际中，并不一定要找到所有的空白位置，如果在开局时就把那些边界的空白位置当作下一步的走法，这样就会导致脱离战场的危险，对于走棋的一方是非常不利的。因此，本系统在走法生成模块中限定了所生成走法的数量，去掉了那些显然不可走的走法，保留了一些好的走法。

首先定义一个函数，来得到规定数量的合法的走法：

```
public static PossibleStep[] GetLimitedPossibleSteps(int[] stepValues, int count,int player)
```

函数包括 3 个参数：

```
int[] stepValues        //两步棋的打分数组
int count               //生成合法走法的数量
int player              //是黑方还是白方走棋
```

函数返回值为一个 PossibleStep 的结构数组，数组的长度为 count，表示了生成合法走法的数量，数组中每一个元素是一种合法的走法，包括某一方走两步棋子。

上文所述的 α-β 剪枝搜索算法就是要在返回的 PossibleStep 结构数组中找到一种最好的走法。

在走法生成模块中，还需要对搜索引擎模块的搜索结果进行分析和处理。

由于六子棋的特殊性，每次走两步棋，那么不得不考虑两步棋的综合效用。但在 α-β 剪枝搜索算法中又必须把两步当作一步处理，这就涉及到一个映射问题。本文采取了把两步棋的棋盘坐标映射到一维空间的方法，用下式计算综合两步的 stepValue：

$$stepValues[((i * 19 + j) * 19 + k) * 19 + l] = tempvalue1 + tempvalue2$$

其中，（i，j）和（k，1）分别表示第一步和第二步的棋盘坐标，tempvalue1 和 tempvalue2 分别表示第一步和第二步的估值，这就解决了“两步”到“一步”的问题。这也就解释了为什么上文把 int[] stepValues 解释为两步棋的打分数组。

这样，走法生成模块便实现了按照六子棋的走法规则生成合理走法将博弈树展开的功能。

下图为已开发的六子棋计算机博弈系统的人机界面：

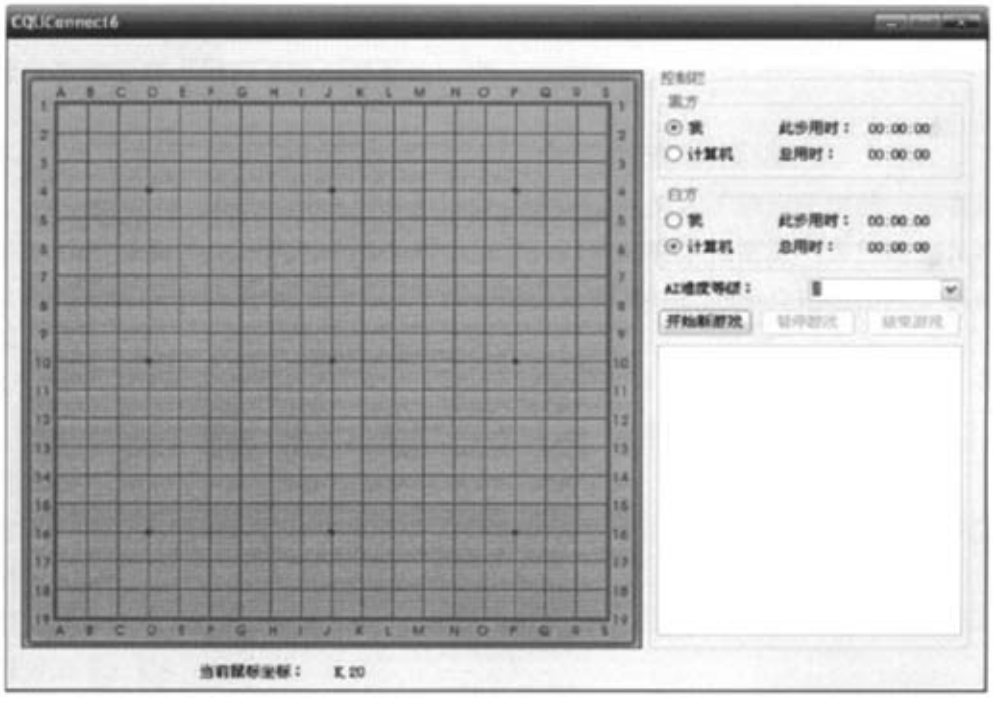


图 4.2 人机界面

Figure4.2 Human machine interface

系统中提供了计时、选择计算机 AI 难度等级、记录历史走棋等功能。

## 4.6 本章小结

本章中提出了六子棋计算机博弈系统平台的构建方法，确定了棋盘的状态空间表示方法，以及在系统的搜索引擎模块中采用了 α-β 剪枝搜索算法并加入了相应的启发式信息，使得搜索的效率和准确性都得到了进一步的提高；在走法生成模块中，采用棋盘坐标映射的方法，解决了六子棋中“两步”到“一步”的综合搜索以及确定落子的问题。但搜索算法以及走法生成的实现都是基于这样的假设，对博弈树中所有节点的打分或者说估值都是准确的，但实际上，这样的估值肯定是有误差的，下一章将详细介绍评估函数怎样来对节点估值的问题。

# 5 基于遗传算法的六子棋计算机博弈系统的评估函数

[本章摘要]本章把研究的重点放在六子棋计算机博弈系统的评估函数模块，根据六子棋棋型特征构建了着子棋力的评估函数，提出了使用遗传算法来对六子棋计算机博弈中评估函数参数进行调整和优化的方法，设计并开发出基于一代遗传操作过程的离线自学习系统。

## 5.1 引言

为了确定评估函数的表达形式，本章首先从六子棋的常见棋型及其状态演变出发，通过提取六子棋的棋型特征，然后构建着子棋力的评估函数，确定了其表达形式和计算方法。

为了进一步提高评估函数的准确性，本章又提出了使用遗传算法对六子棋评估函数的参数进行调整和优化的方法，并且为了加强遗传算法的局部搜索能力，在进化过程中采用了改进的遗传算法——自适应遗传算法。同时，在适应度函数的计算上采用了锦标赛算法。[39]

## 5.2 六子棋的常见棋型及其状态演变的形式化描述

要确定评估函数的表达形式和计算方法，必须深入了解六子棋的一些常用基本棋型以及对六子棋的常用基本棋型之间的内部联系做更深一步的研究。本文把棋型之间的演变情况作为研究其内部联系的基础。下面分别给出六子棋的常用基本棋型以及基本棋型间的状态演变。

### 5.2.1 六子棋的棋型

现给出六子棋常见棋型的定义、英文名称、符号表示和示例如下：

〖六连〗

定义：在棋盘的纵向、横向或斜向的任意一条线上，形成的 6 颗同色棋子不间隔地相连。

英文名称：Continuous six

符号表示：C6

示例：如图 5.1 所示

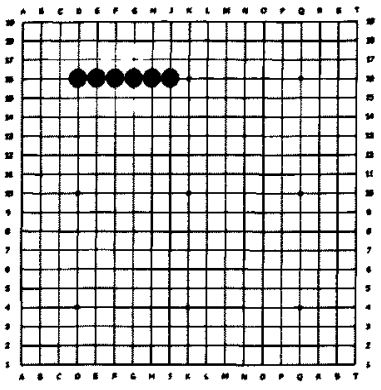

图 5.1 六连
Figure5.1 Continuous six

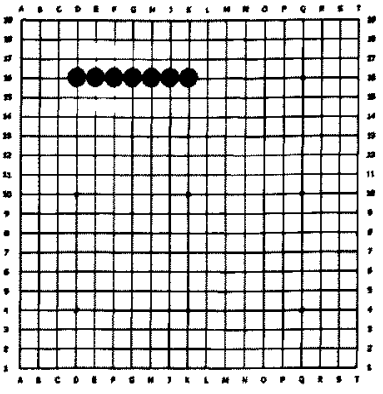

图 5.2 长连
Figure5.2 Continuous long

〖长连〗

定义：在棋盘的纵向、横向或斜向的任意一条线上，形成的 7 颗或 7 颗以上同色棋子不间隔地相连。

英文名称：Continuous long

符号表示：Cl

示例：如图 5.2 所示

〖活五〗

定义：在同一直线上的 5 颗同色棋子，符合“对方必须用两手棋才能挡住”的条件。“挡住”是指不让另一方形成六连或长连。

英文名称：Active five

符号表示：A5

示例：如图 5.3、图 5.4 所示

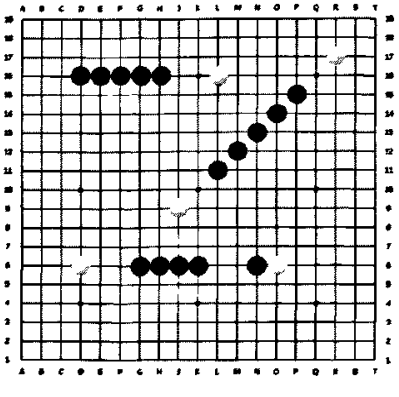

图 5.3 活五
Figure5.3 Active five

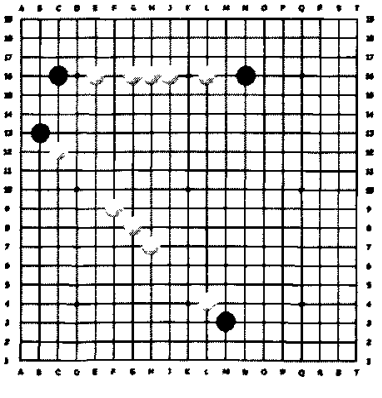

图 5.4 活五
Figure5.4 Active five

〖眠五〗

定义：在同一直线上的 5 颗同色棋子，符合“对方用一手棋就能挡住”的条件。

英文名称：Sleep five

符号表示：S5

示例：如图 5.5 所示

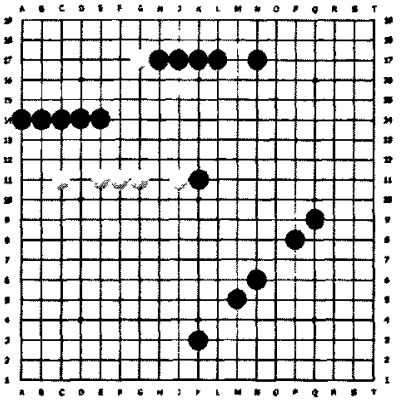

图 5.5 眠五

Figure5.5 Sleep five

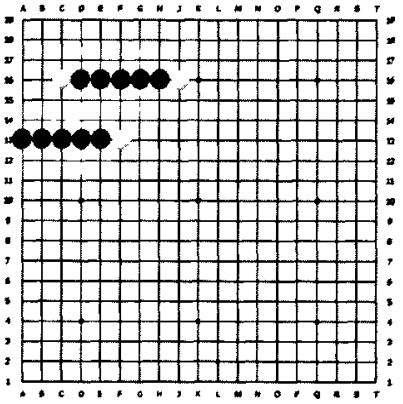

图 5.6 死五

Figure5.6 Dead five

〖死五〗

定义：在同一直线上的 5 颗同色棋子，它们已无法形成六连或长连。

英文名称：Dead five

符号表示：D5

示例：如图 5.6 所示

〖活四〗

定义：在同一直线上的 4 颗同色棋子，符合“对方必须用两手棋才能挡住”的条件。

英文名称：Active four

符号表示：A4

示例：如图 5.7 所示

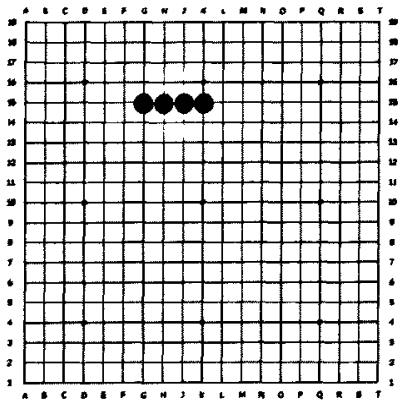

图 5.7 活四

Figure5.7 Active four

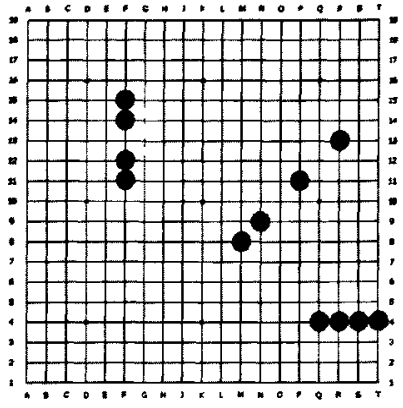

图 5.8 眠四

Figure5.8 Sleep four

〖眠四〗

定义：在同一直线上的 4 颗同色棋子，符合“对方用一手棋就能挡住”的条件。

英文名称：Sleep four

符号表示：S4

示例：如图 5.8 所示

〖死四〗

定义：在同一直线上的 4 颗同色棋子，它们已无法形成六连或长连。

英文名称：Dead four

符号表示：D4

示例：如图 5.9 所示

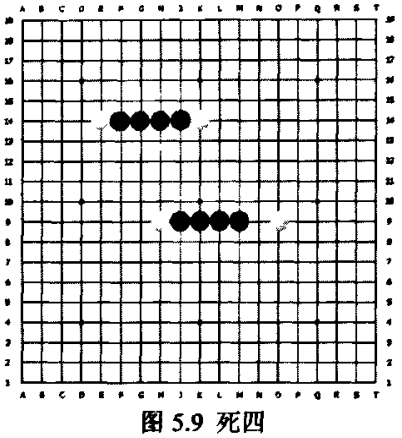

图 5.9 死四

Figure5.9 Dead four

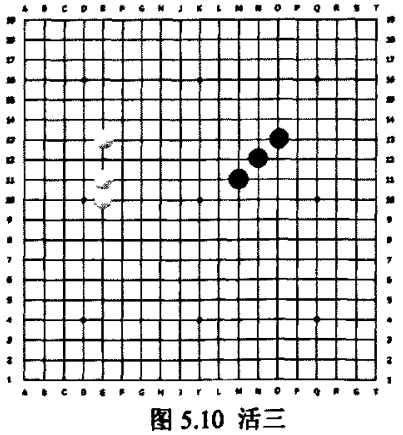

图 5.10 活三

Figure5.10 Active three

〖活三〗

定义：在同一直线上的 3 颗同色棋子，符合“再下一手棋就能形成活四”的条件。

英文名称：Active three

符号表示：A3

示例：如图 5.10 所示

〖朦胧三〗

定义：在同一直线上的 3 颗同色棋子，符合“再下一手棋只能形成眠四，但如果再下两手棋的话就能形成活五”的条件。

英文名称：Indistinct three

符号表示：I3

示例：如图 5.11 所示

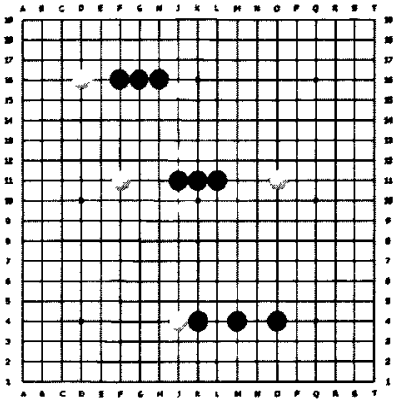

图 5.11 朦胧三

Figure5.11 Indistinct three

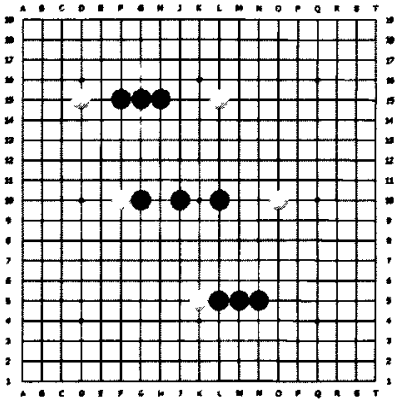

图 5.12 眠三

Figure5.12 Sleep three

〖眠三〗

定义：在同一直线上的 3 颗同色棋子，符合“再下两手棋也只能形成眠五”的条件。

英文名称：Sleep three

符号表示：S3

示例：如图 5.12 所示

〖死三〗

定义：在同一直线上的 3 颗同色棋子，它们已无法形成眠四或者活四了。

英文名称：Dead three

符号表示：D3

示例：如图 5.13 所示

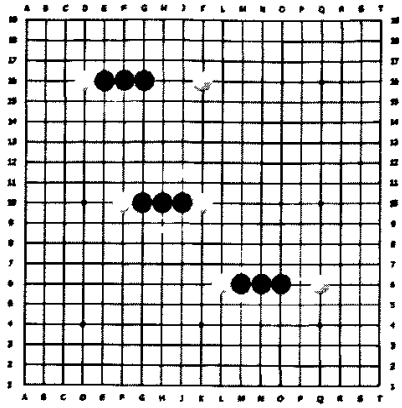

图 5.13 死三

Figure5.13 Dead three

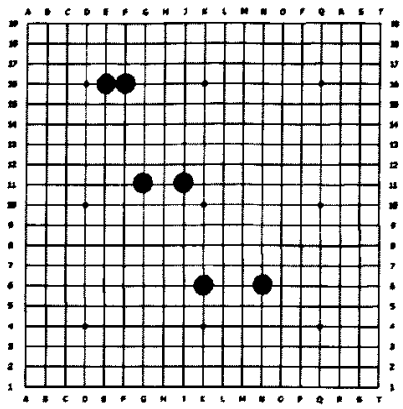

图 5.14 活二

Figure5.14 Active two

〖活二〗

定义：在同一直线上的 2 颗同色棋子，符合“再下两手棋就能形成活四”的条件。

英文名称：Active two

符号表示：A2

示例：如图 5.14 所示

〖眠二〗

定义：在同一直线上的 2 颗同色棋子，符合“再下两手棋只能形成眠四”的条件。

英文名称：Sleep two

符号表示：S2

示例：如图 5.15 所示

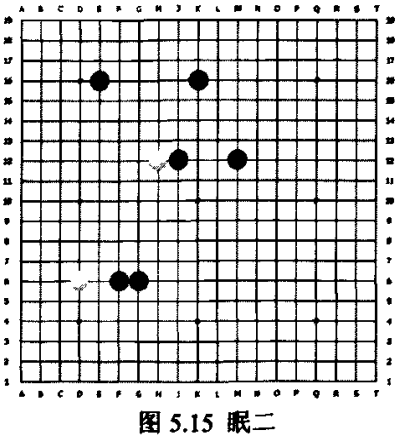

图 5.15 眠二

Figure5.15 Sleep two

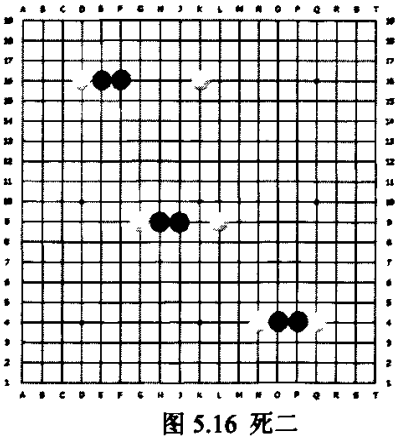

图 5.16 死二

Figure5.16 Dead two

〖死二〗

定义：在同一直线上的 2 颗同色棋子，它们已无法形成眠四或者活四了。
英文名称：Dead two
符号表示：D2
示例：如图 5.16 所示

### 5.2.2 六子棋各棋型间的状态演变

上文已经对六子棋的常用基本棋型做出了简单的定义并举例说明，但这里需要指出的是，对于例子中的每一种基本棋型都例举了不同的对应状态，但并不包含所有可能的对应状态，每种棋型都包含很多不同的对应状态。下面给出六子棋中各棋型的状态演变情况：

A5→（S5∪C6∪A4∪S4）
S5→（C6∪A5∪A4∪A3∪S5∪D5∪D4∪S3∪I3∪D3∪S2∪D2）
D5→（S3∪S4∪D3∪D2）
A4→（A5∪S4）
S4→（S5∪A5∪A3∪I3∪A2）
D4→（D5∪D3∪S3∪S2∪I3）
A3→（A4∪S4∪S3∪D3∪S2）
I3→（S4∪S3∪D3∪S2∪D2）
S3→（S4∪D4∪D3∪D2）
D3→（D4）
A2→（A3∪S2）
S2→（S3∪D2）
D2→（D3∪A2∪S2）

其中，符号“∪”表示“或”的关系。上述的棋型状态演变情况包括了本方走一步和对方走一步时的情况，例如对于 A4→（A5∪S4）的棋型状态演变，假如黑棋当前处于 A4 棋型状态，当在适当的位置落下一颗黑棋，会使黑棋处于 A5 棋型状态，实现 A4→A5 的棋型状态演变；而当在适当的位置落下一颗白棋，会使黑棋处于 S4 棋型状态，实现 A4→S4 的棋型状态演变。并且每一种棋型到另一棋型的演变情况包括不止一种的对应状态的演变，因为每种棋型都包含不同的对应状态。当然棋型演变为自身的情况都存在。例如，A5→A5 这种情况是肯定存在的，但没有给出。

同样，A5→（S5∪C6∪A4∪S4）表示黑白双方的某一方在现有 A5 棋型的情况下，当己方或对方落下一子后，会转变成 S5 或 C6 或 A4 或 S4 的棋型。当

然，每一种棋型包含不同的对应状态。以 A5→S5 棋型状态演变为例，其下包含了棋型对应的不同状态的演变，当然，它们都属于 A5→S5。

## 5.3 六子棋评估函数的确定

评估函数是模式识别和智能算法应用最为广泛的领域。不管多么复杂的评估函数，都可以表示为一个多项式。评估函数一般来说必须包括 5 个方面的要素，分别是固定子力值、棋子位置值、棋子灵活度值、威胁与保护值、动态调整值，每一方面的值又是由许多参数值构成的。即使最简单的评估函数也有 20 多个参数，将这些值线性地组合在一起得到最终的评估值。评估函数包含以上所述 5 方面要素的一般是像中国象棋、国际象棋这类包含不同棋子的棋类，在很多中国象棋以及国际象棋的计算机博弈研究中，其评估函数都包含了这 5 方面的内容。然而对于六子棋而言，它无子与子的差别（对于某一方来说），不像中国象棋那样有“车”“马”“炮”的不同，所以其评估函数只跟当前棋子的落子位置以及其周边棋子的位置状态有关。所以六子棋的评估函数只是以上所述 5 要素的一个子集，因而可以得到简化。

既然说六子棋的评估函数只跟当前棋子的落子位置以及其周边棋子的位置状态有关，那么本系统定义如下函数，来得到当前落子的评估值：

```
public static int GetStepValue(StepStatus[][] chessBoardStatus, Step step)
```

函数包括 2 个参数：

```
StepStatus[][] chessBoardStatus        //当前棋盘状态
Step step                              //当前棋子的落子位置
```

函数返回值为当前落子的评估值。

那么函数内部究竟怎么实现呢？上文已经提到，在当前棋子落子后，实现了某种棋型演变情况下的某种对应状态的演变，我们把这种对应状态的演变提取为一个棋型特征。那么所有棋型之间演变情况的不同对应状态的演变就形成了很多不同的棋型特征，分别给不同的棋型特征打分估值。找出当前棋子落子后所对应的棋型特征，然后根据棋型特征所对应的估值，就能计算出当前落子的评估值。

由于六子棋规则中，在横向、纵向、斜向任一方向上形成 6 颗同色棋子不间隔地相连都为获胜，那么在寻找棋型特征时，横向、纵向、左斜向、右斜向四个方向都要寻找。当然，当前落子的评估值就是其四个方向上棋型特征分别对应的评估值的和。

那么怎样来得到当前棋子落子后所对应的棋型特征呢？本系统定义了四个棋型特征的提取函数，分别来获取当前落子四个方向上所对应的棋型特征：

public static int[] GetHorizonalCodeStatus(StepStatus[][] chessBoardStatus, Step step)

public static int[] GetVerticalCodeStatus(StepStatus[][] chessBoardStatus, Step step)

public static int[] GetLeftSlopeCodeStatus(StepStatus[][] chessBoardStatus, Step step)

public static int[] GetRightSlopeCodeStatus(StepStatus[][] chessBoardStatus, Step step)

每个函数都包括 2 个参数：

StepStatus[][] chessBoardStatus　　//当前棋盘状态

Step step　　//当前棋子的落子位置

函数的返回值分别为横向、纵向、左斜向、右斜向四个方向上的棋型特征，每个棋型特征都是一个整型的状态数组。

首先，从当前棋子的落子位置和当前的棋盘状态出发，以当前棋子的落子位置为中心，分别从横向、纵向、左斜向、右斜向四个方向定长地扫描步长为 13 的点，得到四个方向的状态数组 horizonalCodeStatus、verticalCodeStatus、leftSlopeCodeStatus、rightSlopeCodeStatus，每个数组的长度为 13，数组中的每一位可取 4 个值，-1、0、1、2，分别表示棋盘外、黑棋、白棋、无子的情况。其中，当前的落子在状态数组的第 7 位。例如一个黑棋 A3→A4 的棋型演变情况下，其中一个对应状态演变所对应的棋型特征的状态数组可表示为{2，2，2，0，0，0，0，2，2，2，2，2，2}。

下面列出一些典型棋型特征的状态数组：

{2，2，0，0，0，0，0，2，2，2，2，2，2}

{2，2，2，2，0，0，0，2，2，2，2，2，2}

{0，1，1，1，1，2，0，2，2，2，2，2，2}

{0，2，1，1，1，1，0，2，2，2，2，2，2}

{2，2，2，2，2，0，0，0，2，2，2，2，2}

{2，2，2，2，0，0，0，0，2，2，2，2，2}

理论上，棋型特征的数量将达到 $10^7$ 个。然而有许多的棋型特征对于棋局的影响很小或几乎没有影响，除去这些棋型特征后，剩下的对棋局有影响的棋型特征大概有 $10^3$ 到 $10^4$ 个。例如状态数组{2，2，2，2，2，2，0，2，-1，-1，-1，-1，-1}所表示的棋型特征就对棋局几乎没有影响。

在系统实现中，为了简化状态数组的表示，定义了一个状态数组的映射函数：

public static int GetCodedCondition(int[] aStepStatus)

函数包括 1 个参数：

int[] aStepStatus　　//某个状态数组

函数的返回值为一个整型数值。

具体的映射方法为：

value += value * 4 + aStepStatus[i] + 1，i=0 to 12

其中，aStepStatus 表示某个状态数组，执行循环过程后最终的 value 就是该状态数组的一个映射值。例如状态数组{2，2，2，0，0，0，0，2，2，2，2，2，2}所映射的 value 值为 910652343。需要指出的是，在这种映射方式下，状态数组和 value 的值是一一对应的，不同的状态数组所映射的 value 值是不一样的。换句话说，value 值就代表了所对应的状态数组。

这里需要提出的一种情况是，如果存在状态数组 A 和 B，如果满足 A[i]=B[12-i]，i=0 to 12，那么称状态数组 A 和 B 为对称状态数组。

很显然，对称状态数组所表示的棋型特征是完全一样的，只是顺序颠倒过来。例如状态数组{2，2，2，0，0，0，0，2，2，2，2，2，2}和{2，2，2，2，2，2，0，0，0，0，2，2，2}为对称状态数组，它们都表示了黑棋 A3→A4 的棋型演变过程，其棋型特征是完全相同的，但它们所映射的 value 值是不同的，分别为 910652343 和 915488343。

不同的状态数组表示了当前棋子落子后在某个方向上不同的状态信息或者说棋型特征，而这个棋型特征则反映了当前棋子落子后对于自己的好坏程度，这个好坏程度可以用 stepValue 来定量的表示。那么 stepValue 就是该状态数组所表示的棋型特征的评估值。

估值的原则应该考虑以下几点：

1．棋型特征对棋局的影响越大，估值应该越高；对棋局的影响越小，估值应该越低。

例如状态数组{2，2，2，0，0，0，0，2，2，2，2，2，2}表示的是黑棋 A3→A4 的棋型演变过程，而状态数组{2，2，2，2，0，0，0，2，2，2，2，2，2}表示的是黑棋 A2→A3 的棋型演变过程。从定性的角度看，前一状态数组所表示的棋型特征的 stepValue 值肯定应大于后一状态数组所表示的棋型特征的 stepValue 值。

2．一些威胁性小的棋型特征，其估值的 4 倍得分也不应该超过一个极有威胁性的棋型特征的估值大小。

例如状态数组{0，1，1，1，1，1，0，2，2，2，2，2，2}所表示的棋型特征的威胁非常大，黑棋是防守白棋的一个眠五，对白棋来说形成了 S5→D5 的棋型演变过程，如果黑棋不落此子，白棋将取得胜利。所以此棋型特征的估值将

大于一个类似于{2，2，2，2，2，0，0，2，2，2，2，2，2}这样的状态数组所表示的棋型特征的 4 倍，该棋型特征黑棋仅仅形成了一个活二，即使四个方向上都出现这样的棋型特征，也不会优先考虑。

3．对于一方取得胜利的棋型特征，将赋予一个非常大的估值。

例如状态数组{1，0，0，0，0，0，0，2，2，2，2，2，2}表示了黑棋获胜的棋型特征，将该棋型特征的 stepValue 值赋予 50000。本系统中其它获胜的棋型特征的 stepValue 值都统一赋为 50000。

4．对于对称状态数组所表示的棋型特征的估值必须一样。

例如状态数组{2，2，2，0，0，0，0，2，2，2，2，2，2}和{2，2，2，2，2，0，0，0，0，2，2，2}为对称状态数组，如果前者所表示的棋型特征的 stepValue 值为 2000 的话，那么后者所表示的棋型特征的 stepValue 值也必须为 2000。

根据以上的估值原则，所有棋型特征的 stepValue 值都先由我们人为的给出，但这依赖于编程者的棋类知识和经验，尽管遵循了上述的估值原则，但给出的值仍然不够准确，所以我们在后面提出了通过遗传算法来对 stepValue 值进行调整和优化的方法。

到此为止，我们已经把状态数组一一映射到 value，并且为状态数组所表示的棋型特征的 stepValue 赋予了评估值。那么，便形成了 value 与 stepValue 的一一对应。这样，就可以把 value 作为“键”，把 stepValue 作为“值”存储在一个哈希表中，表的实际长度为 $10^3$ 到 $10^4$，下表中列出了一个典型的哈希表的其中一部分：

表 5.1 哈希表

Table5.1 Hashtable

| value | stepValue |
|---|---|
| 549324218 | 50000 |
| 891121093 | 3000 |
| 910652343 | 2000 |
| 914558593 | 1000 |
| 366277343 | 5000 |
| 415027343 | 5000 |
| 915333593 | 1000 |
| 914552343 | 2000 |
| 915332343 | 2000 |

然后查找哈希表，通过四个方向的 value 分别得到四个方向的 stepValue：stepHorizonalValue、stepVerticalValue、stepLeftSlopeValue、stepRightSlopeValue。

最后，当前落子的评估值可由如下的式子得到：

stepValue = stepHorizonalValue + stepVerticalValue + stepLeftSlopeValue + stepRightSlopeValue。

需要指出的是，哈希表中所存储的 value 值都是或多或少对棋局有影响的值。如果所查找的 value 值在哈希表中不存在，其对应的 stepValue 值默认为 0，表明此时的 value 值所表示的棋型特征对于棋局的影响很小或几乎没有影响。

这样便从当前棋子的落子位置和当前棋盘状态的信息中，通过扫描、查找等操作而确定了当前落子的评估值。

本文在六子棋计算机博弈系统的评估函数模块的设计与实现中用到了哈希表，哈希表是采用“键”、“值”对应的存储方式，通过“键”查找“值”。而“键”一般都是一个整型数值，这样有利于提高查找速度，节省时间，提高效率，所以本文在程序实现中要把状态数组映射为一个整型数值 value。

现已根据六子棋棋型特征构建出着子棋力的评估函数，确定了六子棋评估函数的表达形式和计算方法。但在初始情况下，由于是我们人为地赋值，所以对其棋型特征的估值是不准确的。在这种情况下，本文提出了用遗传算法对六子棋评估函数参数进行调整和优化的方法。

## 5.4 遗传算法[40] [41]

与人工神经网络、模拟退火算法、拉格朗日松弛算法一起，遗传算法(Genetic Algorithm，GA)已经成为解决现代非线性优化问题的一种重要方法，也是近些年发展起来的一种崭新的全局优化算法,由密执安大学的 John Holland 教授首次提出。它借用了生物遗传学的观点，通过选择、遗传、变异等作用机制，实现个体适应性的提高，体现了自然界中“物竞天择、适者生存”的进化过程。遗传算法吸引了大批的研究者，迅速推广到优化、搜索、机器学习等方面。

遗传算法是模拟生物进化的计算机算法，它模拟的是群体的集体进化行为，其中群体中的每个个体表示问题搜索空间中一个可行解。遗传算法是从任意初始的解群体出发，通过群体中个体基因的遗传和变异，从而有效地达到一种稳定的优化状态的繁殖和选择的过程，可使群体进化到搜索空间越来越好的区域。它运用随机而非确定性的规则对一组而非一个点进行全局而非局部的搜索，它利用目标函数而不要求其导数或其他附加限制，它虽然在特定问题上效率也许不是最高，但总体效率远高于传统随机算法，是一种普遍适用于各种问题的简单而又有效的搜索方法。

遗传算法运行过程是典型的迭代过程，其必要工作内容和基本步骤如下：

① 确定合适的编码方案，将解空间与编码空间对应；

② 根据规则定义适应度函数；

③ 制定遗传策略，主要是遗传算子的选择和遗传参数的确定；

④ 随机产生初始群体；

⑤ 以编码来计算群体中个体的适应度，以此判断求解方向；

⑥ 以确定好的遗传算子和运行参数来产生新个体，对群体性能进行判断，以此来决定算法运行与否。

一个基本的遗传算法可以用下面的伪码来描述：

```
Procedure Genetic Algorithm
Begin
  K = 0;   {K 为群体代数}
  初始化 P(k);   {p(k)为第 k 代群体}
  计算 P(k)的适应值;
  While{不满足停止准则}do
    Begin
    K = K + 1;
    从 P(k - 1)中选择 p(k);   {复制算子作用}
    重组 P(k);   {杂交和变异算子作用}
    计算 P(k)的适应值;
  End
End
```

从上面可以看到，遗传算法是个迭代过程，在迭代中，算法保持一个定常规模的群体，每一迭代步(称为步)包括计算出当前群体中个体的适应值，以及基于适应值形成一个新的解群体。

上文已经对遗传算法作了简单的介绍，而传统的爬山法是通过对参数小范围的试探来寻求最优解，并且受初始采样值取值范围以及单方向搜索的限制，很容易陷入局部最优。

模拟退火是一种基于蒙特卡罗迭代求解的启发式随机搜索算法，它试图通过模拟高温物体退火过程的方法，寻找优化问题的全局最优解或近似全局最优解。它可以避开局部最优点，但是致命的缺点是速度太慢,运行时间太长。

相比之下，遗传算法同时使用多组搜索信息，具有很好的全局搜索能力，将重点集中在性能较高的地方，具有很快的搜索速度；而且遗传算法的鲁棒性也

明显优于前两种算法，很有可能在六子棋中取得成功。

由于上文所谈到的六子棋评估函数中，当前落子的 value 值所对应的 stepValue 值是由我们人为地给出，但这很大程度上依赖于编程者的棋类知识和经验，即使遵循了上述的估值原则，也不能保证 stepValue 的值一定很准确。如果通过手动调整大概 $10^3$ 到 $10^4$ 个的 stepValue 值，显得非常困难，很难达到全局最优。所以我们提出了将遗传算法引入到六子棋的评估函数中，来做参数的调整与优化的方法。对参数的调整与优化即是对 stepValue 值的调整与优化。

## 5.5 遗传算法和六子棋评估函数的对应关系及计算机表示[42] [43]

为了把遗传算法引入到六子棋的评估函数中，必须解决的问题是找出遗传算法和六子棋评估函数的对应关系及它们在计算机中的表示方法，怎样把遗传算法与六子棋评估函数相结合，怎样用遗传算法来对六子棋评估函数的参数做调整和优化并在计算机中表示出来，这都是我们需要考虑的问题。

遗传算法处理的对象主要是个体，个体包括一组染色体串，其中每一个染色体对应于评估函数中的一个参数值，也就是上文提到的某一个状态数组的 stepValue，而状态数组所映射的 value 可被看作为参数名,那么存储 value 以及 value 所对应的 stepValue 的哈希表就可以被看作是遗传算法处理的对象——个体。

现在，遗传算法和六子棋评估函数的对应关系及它们在计算机中的表示方法可总结为下表所示：

表 5.2 遗传算法和六子棋评估函数的对应关系及计算机表示

Table5.2 Relationship of GA with Connect6 evaluation function and representation in computer

| 遗传算法 | 评估函数 | 计算机表示 |
|---|---|---|
| 个体 | 评估函数 | 哈希表 |
| 染色体名称 | 评估函数中的参数名 | 哈希表中的“键”value |
| 染色体值 | 评估函数中的参数值 | 哈希表中的“值”stepValue |

有了这样的对应关系后，个体所包含的一组染色体串就代表了参数组的值，也就是所有可能的重要的或对棋局有影响的状态数组的 stepValue。遗传算法就用来调整和优化哈希表中一系列的 stepValue 值。

而每个染色体都是一个基因数组，基因的长短决定编码的精度。现要对每个染色体进行编码，即对 stepValue 进行编码。至于染色体的编码方式，本文选用的是通用的二进制编码方式，即把 stepValue 的值编码为二进制。由于对解的

精度要求不高，二进制的编码和解码都可以快速简单地实现，而且用二进制编码方式进行交叉变异操作时也比其他编码方式更加简捷方便。

由于 stepValue 的初始值是由我们人为给定的一个整数值，所以我们采用了直接编码的方式，也就是直接把 stepValue 的值转化为二进制，编码的精度为 16 位，占 2 个字节，高位用“0”填满。

例如，stepValue = 1000，其基因数组的编码为 0000001111101000；stepValue = 1024，基因数组编码为 0000010000000000；stepValue = 511，基因数组编码为 0000000111111111。

同理，把基因数组的编码解码到 stepValue 也是一样，把二进制编码所对应的整数计算出来即可。

例如，基因数组编码为 0000001100111001，其 stepValue = 825。

到目前为止，我们已经明确了遗传算法和六子棋评估函数的对应关系以及计算机表示。现在我们将用遗传算法来对六子棋评估函数参数进行调整与优化，提出了一种进化一代的实现方法，包括适应度函数的计算、遗传操作过程和改进的自适应遗传算法，并通过一个例子来说明了这种方法的实现过程。

## 5.6 适应度函数的计算——锦标赛算法

众所周知，遗传算法在进化搜索中基本不需要外部信息，仅以适应度函数(fitness function)为依据,利用种群中每个个体的适应度函数来进行搜索。一般而言，适应度函数是由目标函数变换而成的。但是六子棋的评估比较复杂，很难像一般优化问题那样找到真实准确的目标函数，因此本文采用了一个专门用于棋类优化问题的适应度函数计算方法——锦标赛算法。

首先，要对我们自己开发的六子棋计算机博弈程序进行拓展，拓展后的程序包含两个评估函数，但是搜索引擎、走法生成都是共用的。两个评估函数通过读取不同的参数组来进行计算机与计算机的对弈，并决出胜负。这样，就可以通过竞赛的方式来确定下棋双方的适应度，也就是参数组的优劣。在相同适应度的初始状态下，赢的一方对适应度进行加运算(奖赏)，输的一方对适应度进行减运算(惩罚)，和棋则不操作。这样就解决了适应度函数不好确定的问题，实现了通过适应度函数来决定个体的优劣程度，体现了自然进化中的优胜劣汰原则。

在这里，每一个哈希表代表不同的个体，并维护一组参数，也就是哈希表中一系列的 stepValue 值。根据上文所述的遗传算法与六子棋评估函数的关系可知，不同的哈希表中有共同的参数名，即 value，其对应的参数值，即 stepValue 可以是不一样的，这也反映了不同个体之间的差异。两个评估函数分别读取不

同的哈希表中的一系列 stepValue 值来进行计算机与计算机的对弈，也就是个体通过相互之间的竞争、比赛来确定适应度的大小，适应度高的个体拥有一组较好的参数值，可以得到遗传的机会；而适应度低的个体拥有一组较差的参数值，在进化过程中就被淘汰掉。

## 5.7 遗传操作过程

本文的所提出的遗传操作过程可以在单独开发的一个基于一代遗传操作的离线自学习系统上完成。它以原始种群为输入，通过锦标赛选择、均匀交叉、变异等遗传操作过程，最终得到下一代种群。下面将详细介绍遗传操作过程的实现方法。

### 5.7.1 锦标赛选择[44]

遗传算法可采用的选择方法很多，有轮盘赌选择法、局部选择法、截断选择法和锦标赛选择法，本文提出的方法将采用锦标赛选择法和精英选择策略。每两个个体之间进行先后手互换的两场比赛，取出适应度最高的一些精英作为下一代的父个体。个体数为 m 的锦标赛模式下，比赛的次数为 m（m - 1）。

显然，锦标赛模式的缺点在于比赛次数过多，以至于速度过慢。为了减少比赛次数，从而加快速度，本文对于标准锦标赛算法做了相应的改进。

首先，将包含 m 个个体的种群随机分成 n 组，分别记为 Group[i] (i = 1,2,3…,n)，每组包含 m/n 个个体。然后分别在各组内再进行锦标赛训练，得到每个组的冠军，分别记为 Champion[i](i = 1,2,3…,n)，其为筛选出的最佳。最后对 n 个冠军相互之间做交叉和变异操作产生 m-n 个新个体，分别记为 NewBody[i]（i = n+1,n+2,…m），以 Champion[i]和 NewBody[i]为个体形成一个个体数为 m 的新种群，即下一代种群。做了这样的优化后，比赛次数变为了 m（m/n – 1）。

以 m =20，n = 5 时为例，在没有改进之前的比赛次数为 380，而在改进后的比赛次数为 60。假设每次比赛的时间一样，那么在保证优胜劣汰原则的基础上，可以节省约 80% 的时间。

下面以进化一代为例，给出锦标赛选择过程：

① 现有 20 个个体的种群，分别记为 Body[1]、Body[2] …Body[20]。

② 把这 20 个个体随机分成 5 组，分别记为 Group[1]、Group[2] …Group[5]，假设分组结果为：

Group[1]：Body[1]、Body[2]、Body[3]、Body[4]

Group[2]：Body[5]、Body[6]、Body[7]、Body[8]

Group[3]：Body[9]、Body[10]、Body[11]、Body[12]

Group[4]：Body[13]、Body[14]、Body[15]、Body[16]
Group[5]：Body[17]、Body[18]、Body[19]、Body[20]

③ 分别在各组内进行锦标赛训练，训练的结果如下表所示：
每个表中的第一列都表示采取先手的个体，第一行都表示采取后手的个体，比赛结果以第一列的个体为准。假设所有个体的初始适应度都为 0，适应度函数用 F 表示。

表 5.3 第一组训练结果
Table5.3 Group one training result

| Group[1] | Body[1] | Body[2] | Body[3] | Body[4] |
|---|---|---|---|---|
| Body[1] | | 负 | 胜 | 胜 |
| Body[2] | 胜 | | 胜 | 胜 |
| Body[3] | 负 | 负 | | 负 |
| Body[4] | 负 | 负 | 胜 | |

根据表 5.3，可计算出第一组中各个个体的适应度为：
F（Body[1]）= 2；　　F（Body[2]）= 6；
F（Body[3]）= -6；　　F（Body[4]）= -2；
那么，第一组的冠军就为适应度函数最高的个体 Body[2]，
Champion[1]= Body[2]

表 5.4 第二组训练结果
Table5.4 Group two training result

| Group[2] | Body[5] | Body[6] | Body[7] | Body[8] |
|---|---|---|---|---|
| Body[5] | | 胜 | 胜 | 胜 |
| Body[6] | 负 | | 胜 | 负 |
| Body[7] | 负 | 负 | | 负 |
| Body[8] | 平 | 胜 | 胜 | |

根据表 5.4，可计算出第二组中各个个体的适应度为：

F（Body[5]）= 5；　　F（Body[6]）= -2；

F（Body[7]）= -6；　　F（Body[8]）= 3；

那么，第二组的冠军就为适应度函数最高的个体 Body[5]，

Champion[2]= Body[5]

表 5.5 第三组训练结果

Table5.5 Group three training result

| Group[3] | Body[9] | Body[10] | Body[11] | Body[12] |
|---|---|---|---|---|
| Body[9] | | 胜 | 负 | 平 |
| Body[10] | 负 | | 负 | 负 |
| Body[11] | 平 | 胜 | | 胜 |
| Body[12] | 平 | 胜 | 负 | |

根据表 5.5，可计算出第三组中各个个体的适应度为：

F（Body[9]）= 1；　　F（Body[10]）= -6；

F（Body[11]）= 5；　　F（Body[12]）= 0；

那么，第三组的冠军就为适应度函数最高的个体 Body[11]，

Champion[3]= Body[11]

表 5.6 第四组训练结果

Table5.6 Group four training result

| Group[4] | Body[13] | Body[14] | Body[15] | Body[16] |
|---|---|---|---|---|
| Body[13] | | 负 | 胜 | 负 |
| Body[14] | 平 | | 平 | 平 |
| Body[15] | 胜 | 平 | | 平 |
| Body[16] | 平 | 胜 | 胜 | |

根据表 5.6，可计算出第四组中各个个体的适应度为：

F（Body[13]）= -2；　　F（Body[14]）= 0；

F（Body[15]）= -1；　　F（Body[16]）= 3；

那么，第四组的冠军就为适应度函数最高的个体 Body[16]，

Champion[4]= Body[16]

表 5.7 第五组训练结果

Table5.7 Group five training result

| Group[5] | Body[17] | Body[18] | Body[19] | Body[20] |
|---|---|---|---|---|
| Body[17] | | 负 | 胜 | 胜 |
| Body[18] | 平 | | 胜 | 胜 |
| Body[19] | 平 | 负 | | 平 |
| Body[20] | 平 | 平 | 胜 | |

根据表 5.7，可计算出第五组中各个个体的适应度为：

F（Body[17]）= 1；　　F（Body[18]）= 4；

F（Body[19]）= -4；　　F（Body[20]）= -1；

那么，第五组的冠军就为适应度函数最高的个体 Body[18]，

Champion[5]= Body[18]

④ 分别得到五组的冠军 Champion[1] 、Champion[2]、 Champion[3]、Champion[4]、Champion[5]为最佳：

Champion[1]= Body[2]　　Champion[2]= Body[5]

Champion[3]= Body[11]　　Champion[4]= Body[16]

Champion[5]= Body[18]

⑤ 下文将对这 5 个冠军相互之间做交叉和变异操作产生 15 个新个体，然后和这 5 个冠军个体一起形成一个个体数为 20 的新种群，即下一代种群。

### 5.7.2 均匀交叉

交叉的方法很多，有单点交叉、多点交叉、顺序交叉、循环交叉等等。为了使交叉在便于操作的同时更加广义化，本文选用了均匀交叉的方法。

首先，以参数之间的间隔点作为潜在的交叉点，实际上就是对两个个体中相同的 value 对应的不同 stepValue 之间的交换，这里 value 可看作交叉点，即两个个体在交叉点上做 stepValue 的交换。

例如，个体 A 和个体 B 需要在 value = 910652343 的交叉点上进行交叉，交叉前个体 A 在交叉点对应的 stepValue 为 2000，个体 B 的 stepValue 为 1950；那么在交叉操作后，个体 A 在交叉点对应的 stepValue 为 1950，个体 B 的 stepValue 为 2000。

那么怎样来确定哪些 value 来作为交叉点呢？这跟交叉率 Pc 有关。

本文采用的均匀交叉方法根据交叉率 Pc 随机地产生与参数个数等长的 0 - 1 掩码，掩码和参数一一对应。在掩码为 1 对应的参数位置就是交叉点，个体之间需要在这里作参数值的交换；而掩码为 0 对应的参数位置，其参数值保持不变。

例如，上述的冠军 Champion[1]即 Body[2]和 Champion[2]即 Body[5]在均匀交叉前的参数片段如下表所示：

表 5.8 均匀交叉前的参数片段

Table5.8 Parameter fragment before uniform crossover

| Body[2] | | Body[5] | |
|---|---|---|---|
| value | stepValue | value | stepValue |
| 891121093 | 3000 | 891121093 | 3100 |
| 910652343 | 2000 | 910652343 | 1950 |
| 914558593 | 1000 | 914558593 | 1050 |
| 366277343 | 5000 | 366277343 | 5075 |
| 415027343 | 5000 | 415027343 | 4950 |
| 915333593 | 1000 | 915333593 | 950 |
| 914552343 | 2000 | 914552343 | 2050 |
| 915332343 | 2000 | 915332343 | 2000 |

如果此参数片段所对应的均匀交叉 0 - 1 掩码片段为 01001010，那么经过均匀交叉后的参数片段就如下表所示：

表 5.9 均匀交叉后的参数片段

Table5.9 Parameter fragment after uniform crossover

| Body[2] | | Body[5] | |
|---|---|---|---|
| value | stepValue | value | stepValue |
| 891121093 | 3000 | 891121093 | 3100 |
| 910652343 | 1950 | 910652343 | 2000 |
| 914558593 | 1000 | 914558593 | 1050 |

（续上表）

| | | | |
|---|---|---|---|
| 366277343 | 5000 | 366277343 | 5075 |
| 415027343 | 4950 | 415027343 | 5000 |
| 915333593 | 1000 | 915333593 | 950 |
| 914552343 | 2050 | 914552343 | 2000 |
| 915332343 | 2000 | 915332343 | 2000 |

### 5.7.3 变异

变异是指以等于变异率 Pm 的概率改变一个或几个基因，对于二进制串来说，就是根据变异率来实现基因的 0 - 1 翻转。

本文首先用变异率 Pm 来确定需要变异的参数位置，所采用的方法与均匀交叉中确定交叉点的方法类似，根据变异率 Pm 随机地产生与参数个数等长的 0 - 1 掩码，掩码和参数一一对应。在掩码为 1 对应的参数位置就是变异点，个体需要在此变异点变异；而掩码为 0 对应的参数位置，则不需要变异。

个体怎样在变异点变异呢？实际上就是在该变异点改变 stepValue 的值，上文已经把 stepValue 编码为 16 位 2 个字节的基因数组。例如 stepValue = 2000，其基因数组编码为 0000011111010000。现在随机地产生一个长度为 5 的 0 - 1 掩码，分别与基因数组编码的低 5 位基因一一对应。在掩码为 1 对应的基因位置实现 0 - 1 翻转；而掩码为 0 对应的基因位置保持基因不变。这样就改变了基因数组编码，从而改变了 stepValue 的值，实现了变异。

例如，上文中 Body[2]在变异前的参数片段为下表所示：

表 5.10 变异前的参数片段

Table5.10 Parameter fragment before variation

| Body[2] | | |
|---|---|---|
| value | stepValue | 基因数组编码 |
| 891121093 | 3000 | 0000101110111000 |
| 910652343 | 1950 | 0000011110011110 |
| 914558593 | 1000 | 0000001111101000 |
| 366277343 | 5000 | 0001001110001000 |
| 415027343 | 4950 | 0001001101010110 |
| 915333593 | 1000 | 0000001111101000 |
| 914552343 | 2050 | 0000100000000010 |
| 915332343 | 2000 | 0000011111010000 |

如果此参数片段所对应的变异 0 - 1 掩码片段为 00100001，5 位 0 - 1 掩码为 10110，那么经过变异后的参数片段就如下表所示：

表 5.11 变异后的参数片段

Table5.11 Parameter fragment after variation

| Body[2] | | |
|---|---|---|
| value | stepValue | 基因数组编码 |
| 891121093 | 3000 | 0000101110111000 |
| 910652343 | 1950 | 0000011110011110 |
| 914558593 | 1022 | 0000001111111110 |
| 366277343 | 5000 | 0001001110001000 |
| 415027343 | 4950 | 0001001101010110 |
| 915333593 | 1000 | 0000001111101000 |
| 914552343 | 2050 | 0000100000000010 |
| 915332343 | 1990 | 0000011111000110 |

如果保持变异 0 - 1 掩码片段不变，仍为 00100001，而 5 位 0 - 1 掩码为 01101 的时候，那么经过变异后的参数片段又如下表所示：

表 5.12 变异后的参数片段

Table5.12 Parameter fragment after variation

| Body[2] | | |
|---|---|---|
| value | stepValue | 基因数组编码 |
| 891121093 | 3000 | 0000101110111000 |
| 910652343 | 1950 | 0000011110011110 |
| 914558593 | 997 | 0000001111100101 |
| 366277343 | 5000 | 0001001110001000 |
| 415027343 | 4950 | 0001001101010110 |
| 915333593 | 1000 | 0000001111101000 |
| 914552343 | 2050 | 0000100000000010 |
| 915332343 | 2013 | 0000011111011101 |

从上图中可以看出，变异点分别在参数片段的 914558593 和 915332343 的两个位置上，变异前其对应的 stepValue 值分别为 1000 和 2000，在第一种情况

下变异后的 stepValue 值分别变为了 1022 和 1990，在第二种情况下变异后的 stepValue 值分别变为了 997 和 2013。其 stepValue 值的变动幅度都不大，这是由于 5 位的 0 - 1 掩码对应于基因数组编码低 5 位的控制效果，使得 stepValue 值的变动范围在-31 到 31 之间。这非常符合实际情况，每次变异都让 stepValue 增加或者减少一个不超过 31 的值，达到一个逐步调整的效果，不会改变原有棋型特征的性质。如果基因的翻转在基因数组编码的高位上，会大大增加 stepValue 值的变动范围。例如，stepValue = 5000 的基因数组编码为 0001001110001000，如果对第 4 位（高位）的"1"进行 0 - 1 翻转，结果会变为 0000001110001000，对应的 stepValue 值就变为 904，和原来的 5000 相比，一下减少了 4096，这已经完全改变了原有棋型特征的性质。

总之，变异是一种局部随机搜索，与选择/交叉算子结合在一起，保证了遗传算法的有效性，使遗传算法具有局部的随机搜索能力。同时使得遗传算法保持种群的多样性，以防止出现非成熟的收敛。

通过上述方法，便可以将 Body[2]、Body[5]、Body[11]、Body[16] 、Body[18] 这 5 个冠军相互之间做均匀交叉和变异操作产生 15 个新个体 NewBody[6]、NewBody[7] ...NewBody[20]，它们与 5 个冠军一起形成一个个体数为 20 的新种群，即下一代种群。然后重复上述遗传操作过程，这样种群就一代一代地进化下去，最终便可以得到理想满意的一个个体或一组个体。

下图为完成基于一代遗传操作过程的离线自学习系统的程序示意图：


图 5.17 离线自学习系统

Figure5.17 Offline self-learning system

上文已经说明过，个体的 value 值和 stepValue 值是存储在一个哈希表中的，但这是对于程序运行时而言，因为这个哈希表是在程序运行时动态创建的。而真正的 value 值和 stepValue 值其实存储在一个 XML 文件中，哈希表在程序中创建后再从这个 XML 文件中把 value 值和 stepValue 值读取出来并存进哈希表中，程序再对哈希表作操作。图 5.17 中的选择数据按扭就是从外存中的 XML 文件读取 value 值和 stepValue 值到程序动态创建的哈希表中，然后按照上述的方法做均匀交叉和变异的遗传操作，最后再把结果输出到外存的 XML 文件中。

XML 文件中的一个“node”节点就对应于一个参数，其子节点“name”就是参数名，对应哈希表的“键”value，子节点“value”就是参数值，对应哈希表的“值”stepValue。如下所示：

```
<node>
    <name>793464843</name>
    <value>50000</value>
</node>
```

```
<node>
    <name>915488283</name>
    <value>50000</value>
</node>
<node>
    <name>891114843</name>
    <value>50000</value>
</node>
```

系统中要求输入的交叉率 *pc* 和变异率 *pm* 则由下文的自适应遗传算法来计算得出。

## 5.8 改进的遗传算法——自适应遗传算法[45]

在整个遗传算法实现的过程中，交叉率 *pc* 和变异率 *pm* 的选择是影响遗传算法行为和性能的关键所在，直接影响算法的效率以及收敛性。

对于交叉率 *pc* 来说，如果 *pc* 越大，新个体产生的速度就越快，然而 *pc* 过大时遗传模式或信息被破坏的可能性也越大，使得具有高适应度的个体结构很快就会被破坏；但是如果 *pc* 过小，会使搜索过程缓慢，以至停滞不前，得不到适应度较高的个体。

对于变异率 *pm*，如果 *pm* 取值过小，就不容易产生新的个体结构；如果 *pm* 取值过大，那么遗传算法就变成了纯粹的随机搜索算法，失去了其本来的意义。

由此，需要折中考虑交叉率 *pc* 和变异率 *pm* 的取值大小问题。目前，还没有通用的一次性确定 *pc* 和 *pm* 的方法。针对不同的优化问题，需要反复通过实验和调试来确定 *pc* 和 *pm*，这是一件非常繁琐的工作。为此，本文将采用改进的遗传算法——自适应遗传算法[9]，*pc* 和 *pm* 能够随着适应度自动改变。

*pc* 和 *pm* 的计算公式如下：

$$pc=\begin{cases} pc1-\dfrac{(pc1-pc2)(f'-f_{avg})}{f_{max}-f_{avg}}, & f'\geq f_{avg} \\ pc1, & f'<f_{avg} \end{cases}$$

$$pm=\begin{cases} pm1-\dfrac{(pm1-pm2)(f_{max}-f)}{f_{max}-f_{avg}}, & f'\geq f_{max} \\ pm1, & f'<f_{max} \end{cases}$$

其中，$pc1=0.9$，$pc2=0.6$，$pm1=0.1$，$pm2=0.001$。$f_{max}$ 为群体中最大的

适应度值；$f_{avg}$为每代群体的平均适应度值；$f'$为要交叉的两个个体中较大的适应度值；$f$为要变异个体的适应度值。

以上文中的例子为例，$f_{max}=6$，$f_{avg}=0$。

对于个体 Body[2]与个体 Body[5]的交叉，由于 F（Body[2]）= 6，F（Body[5]）= 5，所以此时的 $f'=6$，由以上的计算公式可得，Body[2]与 Body[5]的交叉率 $pc=0.6$；

对于个体 Body[2]的变异，由于 F（Body[2]）= 6，所以此时的 $f=6$，由以上的计算公式可得，Body[2]的变异率 $pm=0.1$。

而对于个体 Body[5]的变异，由于 F（Body[5]）= 5，所以此时的 $f=5$，计算得到 Body[5]的变异率 $pm=0.0835$。

这种经过改进的自适应的遗传算法中的 $pc$ 和 $pm$ 能够提供相对某个解的最佳 $pc$ 和 $pm$，在保持群体多样性的同时，保证了遗传算法的效率和收敛性。

## 5.9 本章小结

本章通过六子棋棋型特征构建了着子棋力的六子棋评估函数，确定了评估函数的表达形式和计算方法。由于人为地根据棋型特征给出的初始评估值不够精确，然后提出了使用遗传算法对评估函数参数进行调整和优化的方法，并设计开发了基于一代遗传操作过程的离线自学习系统，但其有效性还需在以后的工作中来证明。

# 6 开局库设计

[本章摘要]本章主要根据六子棋的常用开局方式，提出了一种动态开局库的设计方法，并分析了此方法的可行性和优越性。

## 6.1 引言

前面的工作已基本上构建起六子棋计算机博弈系统的平台，包括搜索引擎、走法生成、评估函数 3 个模块。系统在没有开局库模块的情况下，已基本能运行，而且具备一定的棋力和智能。本文的第二章已经对六子棋定石，即常用的开局方式作了介绍。本章将根据这些常用开局方式，提出一种动态开局库的设计方法。到目前为止，尽管开局库还并为实现，但提出这种动态开局库的设计方法有助于以后研究工作的继续。

## 6.2 开局库的设计

开局库是独立于搜索引擎、走法生成以及评估函数模块之外的模块，开局库中存储了大量的专家棋谱，如果根节点的局面在开局库中可以查找到，那么就可以提取开局库的对应走法而不必展开博弈树。这样可以避免在开局时由于搜索深度的不足而带来战略上的失误，同时也大大提高了对战的效率。[21]

本文所提出的开局库的设计方法是一种动态的开局库。首先根据常用的开局方式，定义开局状态的集合，这里指黑棋走一颗棋、白棋走两颗棋时的状态，那么集合中的元素就是常见的 20 种开局方式。然后以每一个元素作为根节点，其对应的常见的下一步（实际为两步，分别为黑棋的 4，5 手）的不同的走法作为叶子节点，生成一棵深度为 2 的树。这样便形成了一个树数为 20 的森林。

对于森林中的一棵树而言，对其叶子节点都给予一个相同的初始评分，如果在以后的对弈中，以其中某个叶子节点所导致的比赛结果为胜利的话，就对该叶子节点评分加 1；导致的比赛结果为平局，则评分不变；导致的比赛结果为负，则评分减 1。那么在每次遇到对应于这棵树的开局时，就选择其评分最高的叶子节点作为下一步（即黑棋的 4，5 手）的应对策略。

对于森林中的其它树，也是按上述方法操作。

这种设计方法的实质是，在开局库中引入了一个评分的机制。对于一个固定的开局方式（前 3 手固定），给黑棋的 4，5 手所对应的不同的叶子节点（即不同的应对策略）评分，并通过比赛的结果来不断地改变这个评分。如果在一次比赛中使用了某个评分最高的叶子节点（应对策略）并赢得了比赛，该节点

评分加 1，仍然为最高的评分节点，那么下次比赛时继续使用这个节点来作为应对策略；如果输掉了比赛，该节点评分减 1，当输到一定场次后，其评分就会小于其它某个节点的评分，那么这时另外那个节点便成了评分最高的节点。在下次比赛时，就会使用另外的那个节点来作为应对策略。

这样的开局库设计是用竞争的方式来确定最好的应对策略，具有自学习的能力，并且也避免了在某种开局方式下选择固定的应对策略。

## 6.3 本章小结

在开局库模块中，提出了一种动态开局库的设计方法，以上的分析已经证明了该方法的可行性和优越性，但还需在今后的实践中来证明其有效性。

# 7 系统评价指标

[本章摘要]本章分别从评估函数的准确度、搜索算法的效率、系统的整体性能等方面给出了六子棋计算机博弈系统的评价指标。

## 7.1 评估函数准确度的评价指标

评估函数的好坏是以其适应度函数的大小来决定的，而适应度函数的大小又是通过评估函数读取不同的参数组来进行比赛的结果确定的。那么参数组的好坏直接决定了评估函数的好坏，本文在评估函数中引入了遗传算法来调整和优化参数组，根据遗传算法的特点，随着遗传代数的增加，其个体的平均适应度会越来越高，参数组会越来越好，那么评估函数也就会越来越准确。

既然评估函数的准确度跟进化代数以及在该代数中的适应度有关，那么，可定义评估函数的准确度如下：

评估函数的准确度 = 进化代数权值 × 在该进化代数中的适应度

其中，进化代数权值不容易人为的确定，但是从定性的角度看，n+1 代的进化代数权值应该大于 n 代的进化代数权值。

## 7.2 搜索算法效率的评价指标

导致比赛失败的搜索对于谈及搜索算法的效率是没有任何意义的。所以要评价搜索算法的效率，前提是要获得比赛的胜利。

再者，不同的评估函数对博弈树中节点的打分不同，当然会影响搜索的结果，那么第二个前提是评估函数是相同的。

还有就上文已经提到搜索的精度越高，即搜索深度越深，宽度越宽，搜索时间就越长，所以再把搜索的精度固定下来。

由此，搜索算法的效率就是在比赛获胜的前提下并且评估函数和搜索精度确定的情况下，平均每步棋（实际为两步）所用的时间。

## 7.3 系统整体性能的评价指标

本文以 3 方面的评价指标来衡量系统的整体性能：

(1) 时间复杂度

系统在比赛中进行棋盘扫描、搜索、走法生成所消耗的时间资源的多少是评价指标之一。

(2) 空间复杂度

系统在比赛中的棋局表示、棋盘表示等数据结构以及搜索算法中的递归调用等所消耗的空间资源的多少是评价指标之二。

(3) 获胜概率

最后，系统在与其他六子棋计算机博弈系统比赛或与人的比赛中所获胜的概率也是系统整体性能的评价指标。

## 7.4 本章小结

本章提出了六子棋计算机博弈系统的评价指标，并通过一系列比赛来对系统进行评测。对评测结果的分析与处理，对于以后系统的改进是很有帮助的。

# 8 结论与展望

[本章摘要]本章总结了本文所完成的主要工作并做出了结论，同时对存在的问题和不足进行了说明，最后对进一步的研究方向进行了展望。

## 8.1 研究工作小结

计算机博弈的研究已经为人工智能领域带来了很多重要的方法和理论，并且产生了广泛的社会影响和学术影响以及大量的研究成果，同时也应用于许多棋类游戏的研究与实现中。

六子棋是最近两年才兴起并发展起来的棋类运动，它已经被越来越多的人所接受，但其计算机博弈的研究还相对较少。

由此，本文以现有的计算机博弈技术理论为基础，同时结合六子棋的特点，借鉴了六子棋的发明者台湾吴毅成教授的相关研究后，提出了六子棋计算机博弈及其系统的研究与实现，把六子棋计算机博弈系统分为四个主要模块：搜索引擎、走法生成、评估函数和开局库。

本文除了开局库模块还未实现外，其他三个模块已基本编程实现，采用了微软的.NET 平台，编程语言为 C# ，并且经过测试，系统已具有一定的棋力和智能。

本文的主要研究内容在搜索引擎模块、走法生成模块以及评估函数模块。而重点在评估函数模块，其主要内容包括了以下几个方面：

一、对六子棋的棋型以及棋型间的状态演变给出了形式化描述。

二、根据六子棋棋型特征构建了着子棋力的评估函数，确定了六子棋评估函数的计算机表达形式和计算方法。

三、提出了使用遗传算法对六子棋评估函数参数进行调整和优化的方法。

四、设计并开发了基于一代遗传操作过程的离线自学习系统。

本文的创新之处有以下几点：

第一、在评估函数模块，根据六子棋棋型特征构建了着子棋力的评估函数，确定了其表达形式和计算方法，提出了使用遗传算法对六子棋评估函数参数进行调整和优化的方法。

第二、在搜索引擎和走法生成模块中，由于六子棋的特殊性，每步走两颗棋，本文采取了把两步棋的棋盘坐标映射到一维空间的方法，解决了“两步”到“一步”的综合搜索以及确定落子的问题。

第三、在开局库模块，尽管开局库还未实现，本文提出了一种动态开局库

的解决方案。动态开局库具有自学习能力，同时也可以避免在某种开局状态下选择固定的应对策略。

第四、在程序实现方面，本文采用了哈希表和 XML 文件作支持。

## 8.2 本系统目前存在的问题和不足

到目前为止，本系统已基本实现，而且具有一定的棋力。但是本系统还存在以下问题和不足：

一、遗传算法对评估函数参数调整和优化的有效性还没得到充分的证明。

二、对两步棋的综合评估值是对两步棋分别评估值的相加，虽然绝大多数情况是如此，但对于一些特殊情况，还必须考虑到相加后的加权或加权后的相加。

三、在搜索引擎模块，每次搜索时做的是全盘扫描，影响了搜索效率。没有对 α-β 剪枝算法做更进一步的优化。

四、还没加入开局库，有可能在开局时造成战略失误，同时也影响了效率。

## 8.3 后续工作

针对上文提出的问题和不足，主要还有以下几方面仍需进一步深入研究：

一、将进一步设计实验并通过大量的工作来验证遗传算法引入到评估函数后的有效性，证明本文所提出方法的可行性。

二、在一些特殊情况下，例如遇到诘棋的情况，对于两步棋的综合评估加入特定的权值。

三、进一步改进和优化搜索算法，加入更多的启发式信息，不用每次都做全盘扫描，从而进一步提高搜索效率。

四、建立并完善本文所提出的动态开局库，来避免在开局时可能造成的战略失误，同时也可以进一步提高对战效率。

## 8.4 结　语

希望本文的研究工作能在理论上和实际应用上为六子棋计算机博弈的发展做出微薄的贡献，或是能给关心和支持六子棋运动的人们带来一些有用的启示。

# 致　谢

本文的研究工作是在我的导师李祖枢教授的精心指导和悉心关怀下完成的，在我的学业和论文的研究工作中无不倾注着李老师辛勤的汗水和心血。在此论文完稿之际，首先向李老师表示崇高的敬意和最诚挚的谢意！在攻读硕士期间，李老师严谨的治学态度、渊博的知识、无私的奉献精神使我深受启迪。短短三年时间，我不仅学到了扎实、宽广的专业知识，也学到了做人的道理。在李老师身上，我看到了老一辈科技工作者严谨、求实、创新的科研精神，这些宝贵的精神财富将使我终身受益。

在本课题的研究过程中，同组的刘朝涛师兄和曹志娟师姐以及张颖师弟提供了多方面的帮助和协作；同时论文的主题思想的形成也是我们六子棋计算机博弈小组集体智慧的结晶。

在多年的学习生活中，除了家人的鼎力支持，还得到了许多领导和老师的热情关心和帮助；在日常学习和生活中，实验室的师兄弟都给予了我很大帮助，在此，向所有关心和帮助过我的领导、老师、同学和朋友表示由衷的谢意！

衷心地感谢在百忙之中评阅论文和参加答辩的各位专家、教授！

李果

二 OO 七年四月 于重庆

# 参考文献


[1] 万翼. 计算机国际象棋博弈系统的研究与实现. 西南交通大学硕士学位论文. 2006.8.

[2] 谷赛. 计算机围棋博弈系统的若干问题研究. 清华大学硕士学位论文. 2004.3.

[3] 董红安. 计算机五子棋博弈系统的研究与实现. 山东师范大学硕士学位论文. 2006.8.

[4] 张维迎. 博弈论与信息经济学. 上海人民出版社. 1996.

[5] 谢识予. 经济博弈论. 复旦大学出版社. 2002.

[6] 六子棋主页. http://www.connect6.org/.

[7] I-Chen Wu, Dei-Yen Huang, and Hsiu-Chen Chang. CONNECT6. Hsinchu, Taiwan.2005.12.

[8] David N. L. Levy, eds. Computer Games. New York: Springer New York Inc, 1988.335-365

[9] 许舜钦. 电脑西洋棋和电脑象棋的回顾与前瞻. 电脑学刊 台湾 1990.3.2 :1-8

[10] 张玉志. 计算机围棋博弈系统 [博士学位论文]. 北京: 中国科学院计算技术研究所.1991

[11] Kierulf, Anders. Smart Game Board: A Workbench for Game-Playing Programs, with Go and Othello as Case Studies: [Ph.D. Thesis No. 9135]. Switzerland: Swiss Federal Institute of Technology (ETH) Z"urich, 1990

[12] 蔡自兴, 徐光祐. 人工智能及应用. 北京: 清华大学初版社, 1996

[13] 陆汝衿. 人工智能, 上册. 科学出版社, 1989.

[14] Nils J. Nilsson, 郑扣根, 庄越挺译, 潘云鹤校. 人工智能. 北京:机械工业出版社,2000.

[15] Nils J. Nilsson. Artificial Intelligence A New Synthesis(英文版). 北京: 机械工业出版社,1999.

[16] Yen S J, Chen J C, Yang T N. Computer Chinese chess[J ]. *ICGA Journal*, 2004, (3):3 - 18.

[17] I-Chen Wu and Dei-Yen Huang. A New Family of *k*-in-a-row Games. Hsinchu, Taiwan.

[18] 六子棋的新中文主页. http://www.connect6.org/web.

[19] Francis Dominic Laram. 博弈编程指南. http://www.gamedev.net.

[20] David Eppstein. Strategy and board game programming, http://wwwl.ics.uci.edu/一eppstein/180a/

[21] Marsland T A. Computer chess and search[D]. Edmonton: University of Alberta, 1991.

[22] J. Schaeffer, Distributed Game-tree Searching, Journal of Parallel and Distributed Computing, Vol. 6, 1989

[23] A.Plaat, J.Schaeffer, W.Pijls, A.de Bruin, Best-first Fixed-depth Minimax Algorithms, Aritificial Intelligence, 1996.

[24] David j. Kruglinski, Scot Wingo&George Shepherd. Programming MicrosoftVisual C++.Sth Edition, Microsoft Press, 1999.

[25] Rivest, R.L.Intelligence, Game Tree Searching by MinMax Approximation. Artificial 1998, Vol. 34, No. 1

[26] G. C. Stockman, A Minimax Algorithm Better than Alphabeta? Aritificial Intelligence, Vol. 12, No. 2, 1979.

[27] H. J. Berliner, C. McConnell, B* Probability Based Search, Artificial Intelligence, Vol. 86, No. 1, 1996.

[28] L. Victor Allis: "Searching for Solutions in Games and Artificial intelligence" PartI. PhDThesis, September 1994, ISBN 90-9007488-0.

[29] L. Victor Allis: "Searching for Solutions in Games and Artificial intelligence" PartII.PhD Thesis, September 1994, ISBN 90-9007488-0.

[30] Martin Schmidt: "Temporal Difference Learning and Chess" Part I. Aarhus University.

[31] Knuth, D.E. and Moore, R.W. (1975). An Analysis of Alpha-Beta Pruning. Artificial Intelligence, Vol. 6, No. 4, pp. 293-326

[32] Knuth D.E. and Moore R.W. (1975). An Analysis of Alpha-Beta Pruning. Artificial Intelligence, Vol. 6, No. 4, pp. 293-326. (15, 160)

[33] T. Anthony Marsland. A review of game-tree pruning. ICCA Journal, March1986, 9(1): 3 — 19.

[34] Hall M.R. and Loeb D.E. (1992). Thoughts on Programming a Diplomat. Heuristic Programming in Artificial Intelligence 3: thethird computer olympiad (eds. H.J. Van denHerik and L.V. Allis), pp. 123-145. Ellis Horwood Ltd, Chichester. (6)

[35] Jonathan Schaeffer. The history heuristic and alpha-beta search enhancements in practice.IEEE Transactions on Pattern Analysis and Machine Intelligence, November 1989, pami-11(1):1203-1212

[36] The history heuristic and alpha-beta search enhancements in practice [Jonathan Schaeffer 1992]

[37] Herik, H.J. van den, and Allis, L.V. (eds.) (1992). Heuristic Programming in Artificial Intelligence 3: the third computer olympiad. Ellis Horwood Ltd., Chichester, England

[38] 王小春. PC 游戏编程[M]. 重庆: 重庆大学出版社, 2002. 1 – 27.

[39] Hsu S C, Tsao K M. Design and implementation of an opening game knowledge2base system for computer Chinese chess[R]. Bulletin of the College of Engineering, N T U, 1991, (53) :75 - 86.

[40] Lorenz D, Markovitch S. Derivative evaluation function learning using genetic operators

[A]. Proceedings of the AAA I Fall Symposium on Games: Planing and Learning [C]. New Carolina, 1993. 106 - 114.

[41] Holland J H. A daptation in nature and artificial system [M]. Ann Arbor: The University of Michigan Press, 1975.

[42] 米凯利维茨 Z. 演化程序遗传算法和数据编码的结合[M]. 北京: 科学出版社, 2000. 20 – 23. (Michalewicz Z. Genetic algorithm + data structure =evolution programs[M]. Beijing: Science Press, 2000. 20 -23.)

[43] 李敏强. 遗传算法的基本理论与应用[M]. 北京: 科学出版社, 2002. 1 – 15. (Li M Q. Genetic algorithm as base theory and application[M]. Beijing: Science Press, 2002. 1 - 15. )

[44] 王小平, 曹立明. 遗传算法理论、应用与软件实现[M]. 西安: 西安交通大学出版社, 2002. 195 – 210. (Wang X P, Cao L M. Genetic algorithm —theory, application and programming implement [M]. XiÃan: XiÃanJiaotong University Press, 2002. 195 - 210. )

[45] Angeline J, Pollack B. Competitive environments evolvebetter solution for complex tasks [A]. Proceedings of the Fifth International Conference on Genetic Algorithms [C]. Urbana2Champaign, 1993. 264 - 270.

# 附 录

## A 作者在攻读硕士学位期间科研工作目录

[1] 六子棋计算机博弈系统：搜索引擎、走法生成、评估函数模块的设计与开发

[2] 遗传操作过程离线自学习系统：设计与开发

## B 作者在攻读硕士学位期间发表论文目录

[1] 李果.《基于遗传算法的六子棋计算机博弈系统评估函数参数优化的研究与实现》. 西南师范大学学报（自然科学版），2007 年 4 月.