六子棋计算机博弈及其系统的研究与优化

重庆大学硕士学位论文

学生姓名：张　颖

指导教师：李祖枢　教　授

专　　业：模式识别与智能系统

学科门类：工　学

重庆大学自动化学院

二 OO 八年四月

Research and Optimization of the Computer-game system of Connect6

A Thesis Submitted to Chongqing University

in Partial Fulfillment of the Requirement for the

Degree of Master of Engineering

by

Zhang Ying

Supervisor: Prof. Li Zhushu

Major: Pattern Recognition and Intelligent System

College of Automation of

Chongqing University , Chongqing, China

April, 2008

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。计算机博弈是人工智能的一个果蝇，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。因此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。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，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迫著(Threats－based)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进一步探索。

本文主要对本课题组前期实现的系统四个主要部分（搜索引擎、走法生成、评估函数和开局库）进行了完善和进一步的优化，同时对功能进行了扩展。走法生成模块中利用棋类的战场策略进行了搜索限制和棋型特征码的提取；搜索引擎模块加入策略启发式信息进行优化，使其更“智能”；评估函数模块中，由于特征码的引入使得采用遗传算法进行优化具有实际可行性；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确实得到了显著的提升。

关键词：六子棋计算机博弈，博弈树，启发式信息，基于策略的 α－β 剪枝算法

ABSTRACT

Computer game is one of the most important and challenge subject of AI area, which contributes many important theories and methods to the research of AI and achieves so many production that wildly influences the society and science. Computer game is the drosophila 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, pattern recognition and intelligence method. So In the past half century, scholars from all over the world poured numerous time and energy into the research of computer game such as Othello、checker、chess、Chinese chess、Go-Moku and I-go, and get a good many of shocking results, for example the world chess game champion Kasparov lost his match in the competition with the super computer "Deep Blue" in 1997, which gave the world a big shock. Now the AI of many other chess games has already reached to the level of world champion.

Comparing with chess like Go-Moku and Chinese chess, which have the mature computer-game algorithm now, there are few researches on connect6, a newly invented chess. Although the connect6 game is now appear in some forum and matches on the Internet, it's players are just people. Professor Yicheng Wu, the inventor of the connect6 has given the fairness problem’s solution and threats-based winning strategy, but there is no more deep public 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 future study on the computer-game of connect6.

Based on the former research, in which we divided the computer-game system of connect6 into four parts: searching engine, move generator, evaluation function and starting steps database, this paper is mainly about the farther optimization of the four parts and the whole system. In move generator parts, we combine heuristic information with classical search algorithm, then we limit the search area based on theory of "breakaway moves", as a result, the efficiency of search process are evidently improved. As for the searching engine part, by adding special strategy into the search progress, the search process is more intelligent. Last, as the import of the characteristic code of chess model, the applying of genetic algorithm on the evaluation function is come into ture. 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, Strategies-Based α-β cut-off Algorithm, Game Tree, Heuristic Information

目　　录

1 绪 论

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，某些棋类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优化，也正是对六子棋计算机博弈技术的一个探索。[1]－[3]

1.2 六子棋计算机博弈的研究意义

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。博弈问题无所不在, 小到孩童的游戏与争论, 大到商家的竞争、各种突发事件(恐怖、灾害) 的应急处理、国家的外交、流血的和不流血的战争, 只要局中的双方主体存在某种利益冲突, 博弈便成为矛盾表现和求解的一种方式。博弈与对策将成为一类智能系统研究的焦点问题。

计算机博弈就是让计算机像人一样从事需要高度智能的博弈活动。人们常常把计算机博弈描述为人工智能的果蝇，通过博弈问题来研究人工智能典型问题，具有以下的优点：

①博弈问题局限在一个小的有典型意义的范围内，研究容易深入。

②博弈问题非常集中的体现了人类的智能，已足以为现实世界提供新的方法和新的模型。

③专家经验容易获取。

④进展可以精确的展示和表现出来，不同方法和模型的优缺点也容易比较。

不完备信息博弈的研究还推动了不精确信息、模糊信息推理和复杂环境决策的研究进程。由于博弈不能避免搜索，而搜索对时间的要求较高，所以，博弈一

直是研究并行算法、并行体系结构的工具。

人们通过博弈的研究还提出了认知心理学上的很多新课题，对认知心理学产生了影响。有些博弈项目为一些新模型与新方法的研究提供了帮助。例如 Backgammon 就对神经网络的发展做出了贡献。

计算机博弈中制定决策和选择决策，与政治、军事、经济、日常生活中的决策有很多类似之处，是一种典型的决策系统，所以它的研究对于建立现实社会的决策支持系统有很强的参考价值，也因此，在国外计算机博弈研究项目经常得到军方和政府机构的支持和赞助。1990 年，美军利用超级计算机对“沙漠风暴”行动进行战略模拟，博弈也成为这场有史以来最成功战争中的高技术明星。

而棋类游戏是对现实生活中各种矛盾、战争的模拟，下棋的双方无时不在调动自己的一切智能, 充分发挥逻辑思维、形象思维和灵感思维的能力。同时它规则简单、易于实现，所以，在人工智能领域始终将棋类的机器博弈作为常用的研究平台之一。以棋类游戏为研究和验证平台，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。

六子棋由台湾吴毅成教授于两年前发明，现正在起步阶段。六子棋也逐渐得到学术界的认可并采纳作为研究平台，2007 中国计算机博弈竞标赛就设立了六子棋比赛项目，国际上，ICGA 也将六子棋作为了比赛棋种。同时也在普通爱好者中间普及，在这期间，已经有好几个站点支持在线的六子棋游戏。

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很少。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。目前关于六子棋的学术资源还是十分稀少，可见的学术资源只有台湾吴教授的相关文章和本实验室的一篇文章。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]。2007 计算机博弈比赛上也涌现出很多优秀的程序，但是明显可以看出程序都还处在起步阶段，各设计者都是在摸索中前进。

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，每手每方走下两颗棋子。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

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[8]－[15]：

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 RS ö6000S 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

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少部分大学在研究，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。使其正在迅速发展，国际上，ICGA 已经把六子棋列入比赛项目；国内，在 2007 年第二届中国计算机博弈竞标赛赛上也第一次设立了六子棋的比赛项目。

目前文献可查的只有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[7]。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[7]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。例如图 1.1 中的(a) 单迫著(one threat), (b) 双迫著(two threats), (c) 三迫著(three threats)。

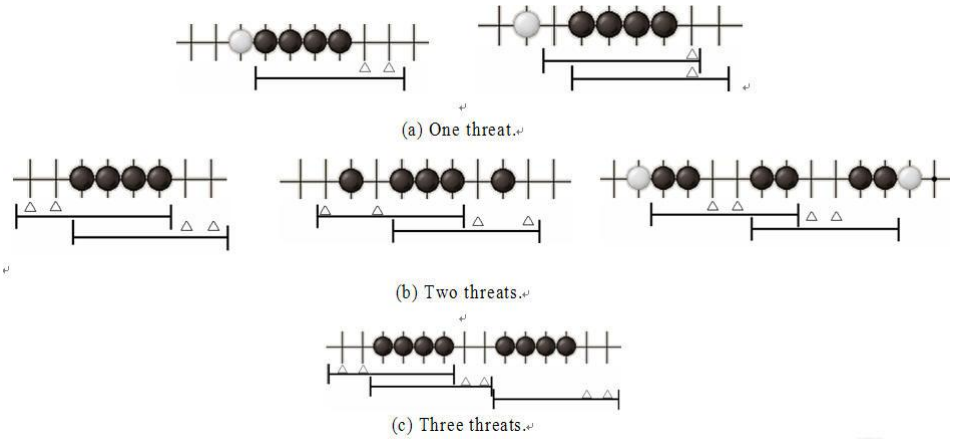

图 1.1 迫著示意图

Fig.1.1 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[7]。

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。我们可以从图 1.2 中看出。

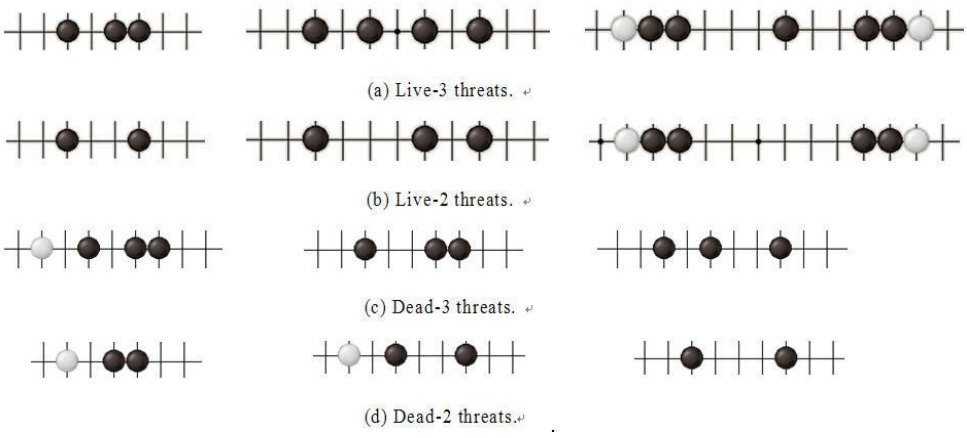

图 1.2 -t 迫著

Fig.1.2 –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

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，六子棋也不例外，这些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。实际中由于计算机能力不可能无限扩展所有局面，所以只有通过有效的搜索算法在有限的时间和扩展内找到正确的解。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。而最佳路径上的第 1 步棋便成为当前局面的最佳着法[32]。

而由于六子棋的特殊性，每手每方走两颗棋子，又必须考虑两步棋的综合效用。因此实际搜索深度就会成级数倍的增长。所以启发信息和有效的搜索方法的结合就对搜索效果十分必要，加入棋类相关的启发式信息（后期最重要）后可以缩小博弈树的规模而避免冗余计算和加快搜索速度，同时还能不损失准确度，甚至提高准确度。因此确定搜索算法后最重要的就是如何去优化这个算法使其达到最佳以及如何准确地提取棋局信息，而棋类的知识十分庞大，所以其优化的余地和信息提取地难度也十分大，这也是几乎所有计算机博弈研究的重点。

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。很多搜索方法都是建立在已经对局面有一个评估的基础上的，两者紧密结合，相辅相辅相成。就像一

个优秀的棋手总有一个良好的对棋局的判断能力，从而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。所以，只有有了良好的评估函数搜索模块才能保证较快地找到正解，该函数的好坏直接影响搜索中的取舍从而决定棋力水平。但要把这一整套的思维物化成一个数值评估函数来评估, 本身就是一个相当复杂的问题。但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。[19]

六子棋计算机博弈系统中评估函数模块优化的任务是：本确定评估函数的参数类型以及参数个数，通过分析棋型特征码，来解决研究和解决“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（时间）

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。

由于六子棋的特殊性，每次一方走下两颗棋子增加了分支因素，其 game-tree 的复杂度可达 10^{140}（按 30 手计算），远大于五子棋，state-space 复杂度可达 10^{172}，与围棋相当[7]。具有高的复杂度同时有着简单的规则，使它成为一个新的研究热点。这也是本文选取 6 子棋作为实现和研究平台的原因。

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。在每个模块中采用结合博弈通用技术和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

六子棋规则简单复杂度高，可以避免把过多的精力放到规则实现上而把主要精力集中到智能实现上，很适合作为一个智能研究平台，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，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，将实现带有启发式信息的 α-β 剪枝搜索算法；在评估函数模块中，针对新的棋型特征构建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

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[7]：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，也是比赛标准棋盘，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[7]。

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[18]：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[7]：

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[18]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[18]：

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[7]。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，如表 2.1 所示：

表 2.1 常见棋类状态空间和博弈树复杂度

Table2.1 State-space and game-tree complexity of common chesses

棋种	状态空间复杂度	博弈树复杂度
国际象棋	10^{46}	10^{123}
中国象棋	10^{48}	10^{150}
围棋	10^{172}	10^{360}
六子棋	10^{172}	10^{226}

2.5 六子棋定石和诘棋

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。图 2.1 所示就是其中比较典型的定石。

诘棋就是一种刻意安排的棋局,而有一些巧妙的攻击胜的必胜手法,以作为题目。就像象棋的残局，围棋的诘棋，常常会有一些意想不到的妙手隐藏其中，这也是加强棋力的重要手段。六子棋自然也有诘棋,如图 2.2。但有些六子棋的诘棋并非只有唯一的一解，对这样的情形，答案尽量选择较简易的解法。

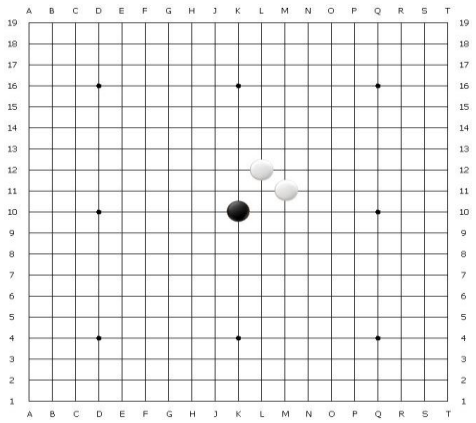

图 2.1 日日局

Fig.2.1 Ri-ri start

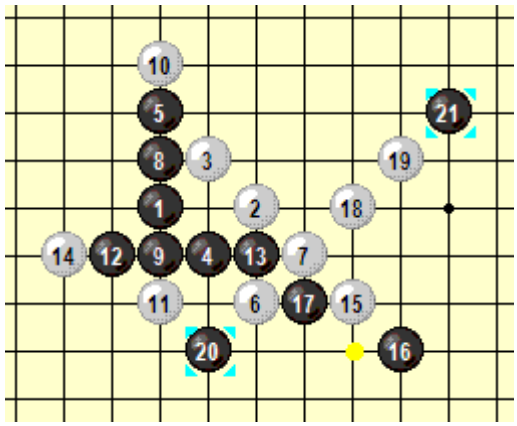

图 2.2 诘棋

Fig.2.2 Tsume-go

2.6 六子棋发展动向

2.6.1 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。

2.6.2 六子棋学术活动

① 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 其他：

在国内，2007 第二届全国计算机博弈锦标赛中六子棋被列入比赛项目。

2.7 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

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。

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 位棋盘[19]

在上世纪 60 年代末诞生，运行于 64 位机上，对于像象棋、西洋跳棋之类在 64 格棋盘上的游戏来说，“64”恰巧是象棋棋盘格子的数目，所以这就有可能让一个字来表示一个棋盘，以判断某个格子的状态是“是”或者“非”。因此，一个完整国际象棋的局面需要用 12 个位棋盘表示：白兵、白车、黑兵等等。再加上两个用来表示“所有白子”和“所有黑子”的位棋盘，以加快运算速度。也可以用一个位棋盘表示被某种子力攻击到的格子，诸如此类，这些位棋盘可以灵活运用在着法产生的运算过程中。

位棋盘之所以有效，是因为很多运算可以转化为处理器的一个逻辑指令。例如，如果在当前的棋盘上，要产生白马的所有着法，那么只要找到当与前位置相关联的“马能走到的格子”的位棋盘，并“与”(AND)上“所有被白方占有的格子”的位

棋盘的补集(就是对这个位棋盘作“非”(NOT)运算)，因为马的着法限制仅仅在于它不能去吃自己的子。

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：

$$B=[b_{i,j}]_{a\times b},\begin{cases}b_{i,j}=1, s_{i,j}\neq 0\\ b_{i,j}=0, s_{i,j}=0\end{cases} \tag{3.1}$$

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。另外，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。

更复杂一些的如文献[32]中“棋天大圣”，还有兵种编码和棋子编码，如表3.1和3.2 。矩阵有兵种状态矩阵S^B_0和棋子状态矩阵S^M_0, 见式3.2，3.3。

表 3.1 象棋兵种编码

Table 3.1 Chess arm coding

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
齐天大圣编码	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
齐天大圣编码	-1	-2	-3	-4	-5	-6	-7

表 3.2 象棋棋子编码

Table 3.2 Chinese Chess chessman coding

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		红象				红兵		

$$S_0^B=\begin{bmatrix} -2 & -3 & -5 & -6 & -1 & -6 & -5 & -3 & -2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -4 & 0 & 0 & 0 & 0 & 0 & -4 & 0 \\ -7 & 0 & -7 & 0 & -7 & 0 & -7 & 0 & -7 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 7 & 0 & 7 & 0 & 7 & 0 & 7 & 0 & 7 \\ 0 & 4 & 0 & 0 & 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 3 & 5 & 6 & 1 & 6 & 5 & 3 & 2 \end{bmatrix} \tag{3.2}$$

$$S_0^M=\begin{bmatrix} 2 & 4 & 10 & 8 & 1 & 9 & 11 & 5 & 3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 6 & 0 & 0 & 0 & 0 & 0 & 7 & 0 \\ 12 & 0 & 13 & 0 & 14 & 0 & 15 & 0 & 16 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 28 & 0 & 29 & 0 & 30 & 0 & 31 & 0 & 32 \\ 0 & 22 & 0 & 0 & 0 & 0 & 0 & 23 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 18 & 20 & 26 & 24 & 17 & 25 & 27 & 21 & 19 \end{bmatrix} \tag{3.3}$$

3.2.3 着法生成

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[19]。

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

以图 3.1 中国象棋为例，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[32]。

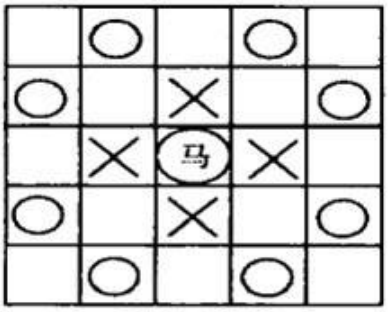

图 3.1 走马匹配模板

Fig.3.1 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。[32]

3.2.4 机器博弈、搜索技术

① 博弈树

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。例如图 3.2 是井字棋(Tic-tac-toe)的博弈树[20]：

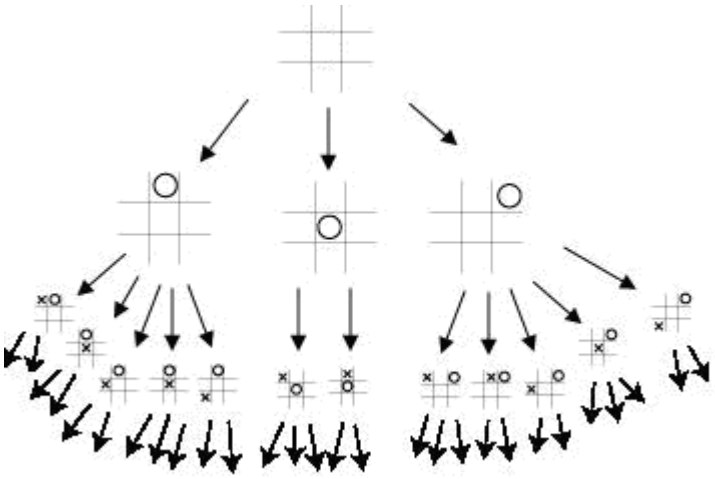

图 3.2 井字棋博弈树

Fig.3.2 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，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。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。[32]

1） 极大极小搜索[22]－[25]

Claude Shannon教授早在1950年首先提出了“极大-极小算法” (Mini-max 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。

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

$$F(s)=m-n \tag{3.4}$$

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3.4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c 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，如图3.3中，A3 走步为根的最佳走步。这就是极大极小搜索过程(MIN-MAX decision)。

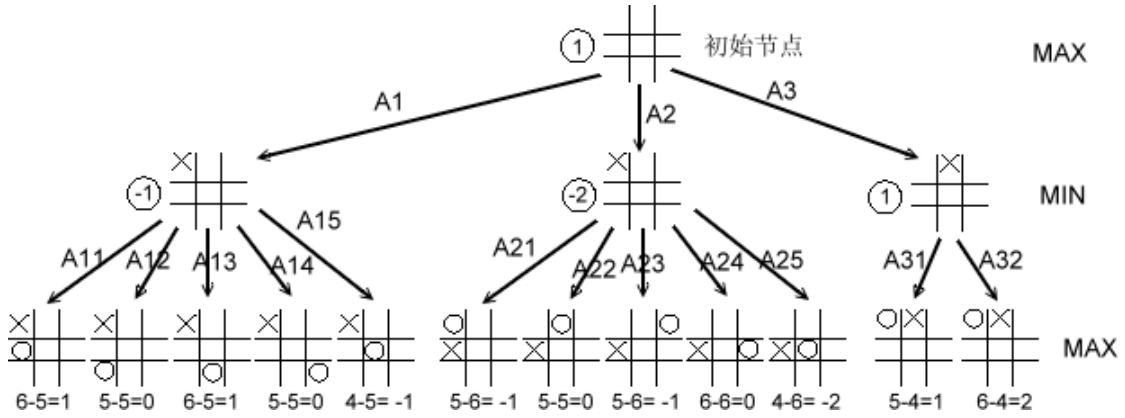

图 3.3 井字棋的极大极小搜索过程

Fig.3.3 Tic-tac-toe MIN-MAX decision

2) 负极大值算法[38]

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。

算法公式：
$$F(v)=\max\{-F(v_1),-F(v_2)...,-F(v_n)\} \tag{3.5}$$

其中，v 1, v 2,…., 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) α-β 剪枝搜索[26]－[35]

前面所述的极大极小搜索算法在使用上十分困难，原因就是搜索量太大，时间太长，而α-β剪枝搜索使极大极小搜索的实现成为了可能，成为很多高级算法的

基础。

剪枝：我们假设五子棋用一个静态估值函数，用 MIN－MAX 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 MIN-MAX 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 MIN-MAX 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。

我们将走棋方定为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着法。

评估函数为： $$value = F(s)$$ （3.6）

其中S为当前棋局状态，value为F返回的评估值

α剪枝模型：图3.4中，从第一个左路分枝的叶节点倒推得到根MAX节点的值5，根节点暂时取该值5，并记α为5。此α值作为MAX方着法指标的下界。在搜索此根MAX节点的其它子节点,例如B分支，B分支的MIN层节点必定≤3，而MAX层节点取其子节点中最大的，而3<5,所以B分支在“3”节点后不用再继续扩展，可以直接剪掉此枝(整个B分支)，从而节省了大量的搜索时间，其他分支与此类似。此类剪枝称为α剪枝。当然α值并不是一直不变地，若某一直孙节点全部比该α值大，则α要重新界定。

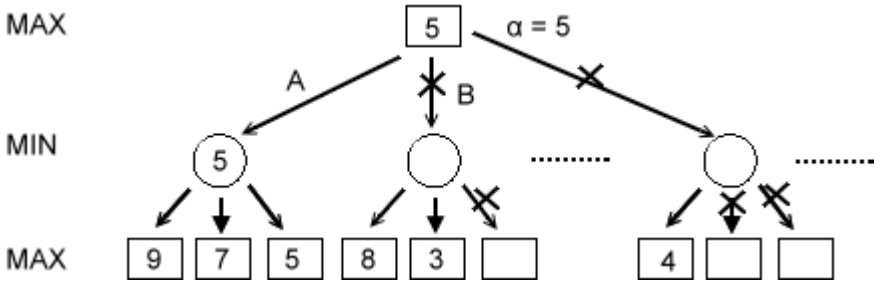

图 3.4 α 剪枝模型

Fig. 3.4 α Pruning model

β剪枝：同理, 如图3.5，由第一左路分枝的叶节点倒推得到根MIN节点的值, 可表示到此为止对对方着法的钳制值,记为β。此值作为MIN方可能实现着法指标的上界。在搜索其它分之孙节点, 如果发现一个孙节点评估值高于上界β（9<11）,则可以剪掉此枝(B)，此类剪枝称为β剪枝。同理，β值也是随时变化的。

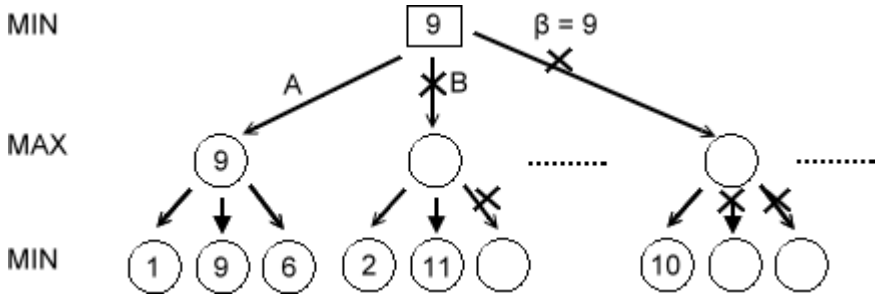

图 3.5 β 剪枝模型

Fig.3.5 β Pruning model

在实际多层搜索中，两者相互配合就实现了 *α-β* 剪枝搜索。需要指出的是，α-β 剪枝虽然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。但该方法一下便为博弈树搜索效率的提高开创了崭新的局面。

设 Alpha 结点下需要搜索的结点数为 Alpha(n)，n 为该结点的深度，那么这个函数的递归表达式是[19]：

Alpha(n) = Alpha(n -1) + (b x 1) x Beta(n - 1) (3.7)

用同样的方法可以构造一个 Beta(n)的函数：

Beta(n) = Alpha(n - 1) (3.8)

把它代入前面的 Alpha(n)函数中(注意，n 要替换成 n - 1，n - 1 要替换成 n - 2)，就可以得到下面的函数：

Alpha(n) = Alpha(n - 1) + (b - 1) x Alpha(n - 2) (3.9)

加上初始了条件 Alpha(0) = Beta(0) = 1(即 Alpha(0) = 1，Alpha(1) = b)后，这个方程是可以解出来的。

α-β剪枝搜索是一种基于α-β剪枝的深度优先搜索。它去掉了一些不影响最终结果的分支而返回与MIN-MAX相同走步的过程。1975年Knuth和Moore给出了α-β算法正确性的数学证明[31]。α-β剪枝算法的效率与子节点扩展的先后顺序相关. 在最理想情况下(极小树），其生成的节点数目为[32]：

$$N_D = 2B^{D/2} - 1 \quad (D \text{ 为偶数}) \tag{3.10}$$

$$N_D = B^{(D+1)/2} + B^{(D-1)/2} \quad (D \text{ 为奇数}) \tag{3.11}$$

其中 D 为搜索深度，B 为分枝因子。在不使用 α-β 剪枝时，需要搜索的节点数是 $N_D = B_D$，即极大树。所以最理想情况下 α-β 算法搜索深度为 D 的节点数相当于搜索深度为 D/2 的不做任何剪枝节点数。

α-β 剪枝算法是通过程序的递归来实现的，其负极大值形式伪代码如下：

```
int AlphaBeta(int depth, int alpha, int beta)
{       if (depth == 0)
```

```
{ return Evaluate(); }
GenerateLegalMoves();
for (PossibleMoves ())
   {   MakeNextMove();
       value = -AlphaBeta(depth - 1, -beta, -alpha);
       UnmakeMove();
       if (val >= beta) {    return beta;    }
       if (val > alpha) {    alpha = val;    }
      }
return alpha; }
```

其中函数递归传递的参数有层数，alpha 值和 beta 值，一般初始化分别取－∞和＋∞。

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-β cut-off) 算法，也取得很好效果。

4) α-β 窗口搜索、期望搜索

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。

比如“期望搜索”[36]，开始时用负无穷大到正无穷大来限定窗口，然后在期望值附近设置小的窗口。如果实际数值恰好落在窗口以内，那么你赢了，你会准确无误地找到路线，并且比其他的路线快(因为很多路线都被截断了)。如果没有，那么算法就失败了，但是这个错误是很容易被检测的(因为“最小-最大”值就是其中一条边界)，此时用一个更大的窗口重新搜索。如果前面的情况比后面的情况多，那么总体上你还是赢了。很明显，预先猜的数值越好，这个技术的收效就越大。

5）MTD(*f*)算法

MTD 算法[36]来源于期望搜索，于 1995 年左右由研究员 Aske Plaat 等人提出。基本思想是：如果把带期望 Alpha-Beta 搜索的窗口大小设定成 0，将会发生什么事？它当然永远不会成功。但是如果它成功了，那速度将是惊人的，因为它把几乎所有的路线全都截断了。现在，如果失败意味着实际数值低于你的估计，那么用稍低点的宽度为零的窗口再试一次，重复下去。这样，就等于用 α-β 搜索来做某个“最小-最大”值的折半查找(实际运作中 MTD(*f*)是以迭代的形式收敛的)，直到你最终找到那个宽度为零的窗口。这个设想的具体实现称为 MTD(*f*)搜索算法。

MTD(*f*)在整个过程中只使用极小窗口，并且每次都从根结点开始的，它只需要传递两个参数(深度 *n* 和试探值 *f*)，这个过程极大程度地依赖于置换表。但加上置换表的运用，MTD(*f*)呈现出惊人的效率，还善于做并行计算。它在“粗糙”(简单且快速)的局面分析中运行得更好，很明显，如果局面评估的最小单位越大，它搜索的步数就越少。

它的实现代码简单，但是效果十分优秀。

```
function MTDF(root : node_type; f : integer; d : integer) : integer;
g := f;
upperbound := +INFINITY;
lowerbound := -INFINITY;
repeat
        if g == lowerbound then beta := g + 1 else beta := g;
          g := AlphaBetaWithMemory(root, beta - 1, beta, d);
        if g < beta then upperbound := g else lowerbound := g;
until lowerbound >= upperbound;
return g;
```

6) 迭代深化搜索[32]

不难想象，深度为 D-1 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$\sqrt{B}$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。

7) 单步延伸

假设在搜索当中某一局面的评估值value＝+10 000，而其他局面的值远远小于这个值，为了节约时间，就会单独的扩展这个高评估值局面。“深蓝”的小组发展了这个思想并提出了“单步延伸”(Singular Extension)的概念。如果在搜索中某步看上去比其他变化好很多，它就会加深这步搜索以确认里边没有陷阱(实际过程远比这里说的要复杂，但基本思想没变)。但单步延伸是耗费时间的，对一个结点增加一层搜索会使搜索树的大小翻一番，评估局面的计算量同时也翻一番。换句话说，只有深蓝那种硬件水平才吃得消它。但是它的成效是不可否认的。所以本文在实现系统上只是借鉴了这个思想，并引入的不是某一步的评估值，而是某一特殊状态来触发单步延伸。

8) 启发式搜索(Heuristic search)

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[12]。其实启发搜索不是一种搜索，而是一类搜索的代表。所有加入启发式信息（领域知识）来优化搜索算法的都可以归类于启发式搜索。

这里值得一提的是，很多算法都是基于α-β搜索，加入各种启发式信息优化而形成了各式各样的很多新的搜索算法。这里对该搜索描述的简单不是它不重要，相反的是它很重要，但是由于启发式信息很多，且因人、棋种领域而异，因此这里不可能一一列出所有的搜索算法，详细的介绍可以参阅各领域和实现平台的相关参考文献。

以上介绍的搜索方法不过是博弈搜索的冰山一脚，博弈树的搜索算法丰富多彩,优化、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

通过上面的介绍，我们知道很多搜索方法都是建立在已经对局面有一个评估的基础上的，两者紧密结合，相辅相辅相成。就像一个优秀的棋手总有一个良好的对棋局的判断能力，从而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。所以，只有有了良好的评估函数搜索模块才能保证较快地找到正解，该函数的好坏直接影响搜索中的取舍从而决定棋力水平。但要把这一整套的思维物化成一个数值评估函数来评估, 本身就是一个相当复杂的问题。

① 整体考虑[20]

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，如图 3.6 所示：

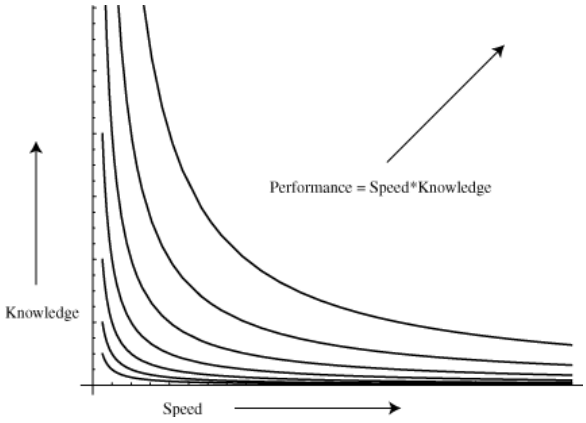

图 3.6 程序质量和知识速度的关系

Fig.3.6 Relationship of the performance with knowledge and speed

② 组合评估要素[19] [20]

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 (3.12)

白方：ws + (1 - bs - ws) * wm + (1 - bs - ws - bm - wm) * we (3.13)

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[20]：

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)：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[20]：

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)。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,由密西根大学的 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。

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)。人工神经网络是利用非线性映射的细想和并行处理的方法采用神经网络本身的结构表达输入和输出关联知识的隐函数编码，输入共建和输出空间的映射关系通过网络结构的不断学习调整，最终以网络结构的特定结构表达。实际上这更多地是一种评估函数的类型，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做出坏的评价，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[16]。但是要获得好的输出需要大量的训练，比如通过大量经典的专家棋谱。

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，后文介绍的 6 子棋平台也是根据这几个模块的划分逐步搭建起来的。

4 六子棋计算机博弈系统的优化

4.1 引言

首先以前文所述计算机博弈通用结构模型为基础设计出简单的 6 子棋计算机博弈平台结构，然后通过定义各种数据结构、搜索算法、评估体系等来实现结构模型中的各个模块功能。在平台的实现过程中就同时伴随着各种测试和优化。之后以平台为基础针对 6 子棋的特点对各个模块进行相应的优化，本章中主要优化对象为前期系统实现时的棋局数据结构、搜索算法，通过对比分析找出适合本平台最优的方法。评估函数的整体优化放到下一章单独介绍。

4.2 六子棋系统的实现结构模型

本文在 6 子棋的实现上在结构上采用了计算机博弈系统设计的通用模型和模块。结构模型和模块如图 4.1 所示。

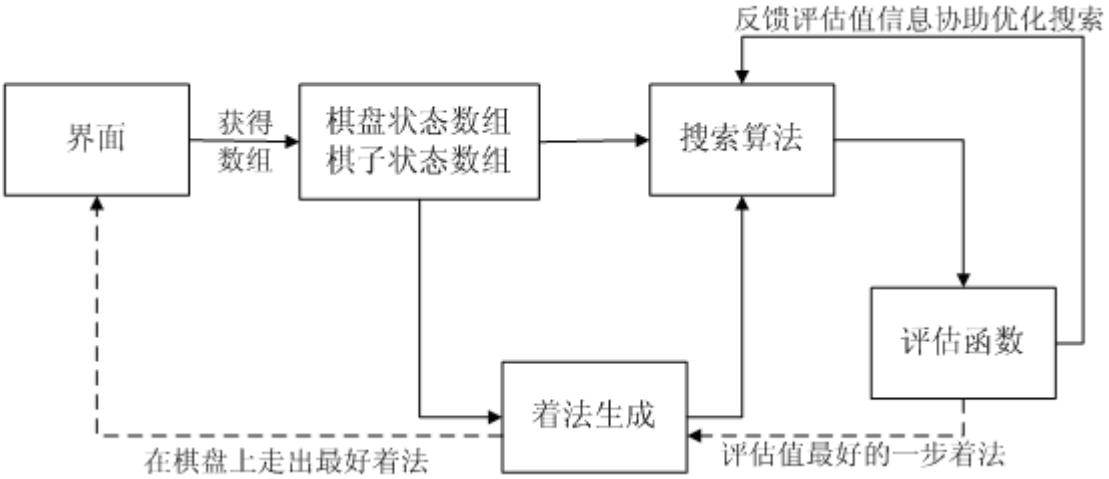

图 4.1 六子棋博弈系统模块及结构模型图

Fig. 4.1 Module and structure model of connect6

4.3 棋局信息的数据表示

要让计算机能下棋，首先就要以计算机能接收的数据格式表示棋盘的相关信息。这些信息就是以后搜索、估值等算法操作的基础，所以信息的准确和完全性就十分重要。我们通过分析六子棋特点结合我们的评估需要的参数，总结出 3 个必须要表示的数据结构：棋局状态信息矩阵，棋子状态矩阵，棋型特征表。

4.3.1 棋盘状态表示

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。本系统定义了一个 19×19 的矩阵 S 来表示 19×19 的棋盘上的棋子信息。其中矩阵中每

个数据对应棋盘上相应位置的棋子信息：黑，白，或者空。然后定义棋子编码，见表 4.1。则可得出任意时刻棋局当前状态的棋盘状态矩阵 S，见式 4.2。

棋盘和矩阵之间的坐标变换公式为（棋盘坐标系如图 4.2 中箭头所示）：

$$a = (x-12)/24$$
$$b = (y-12)/24 \tag{4.1}$$

其中(x,y)为界面棋盘上当前子的坐标，[a,b]为矩阵中的数据位置。

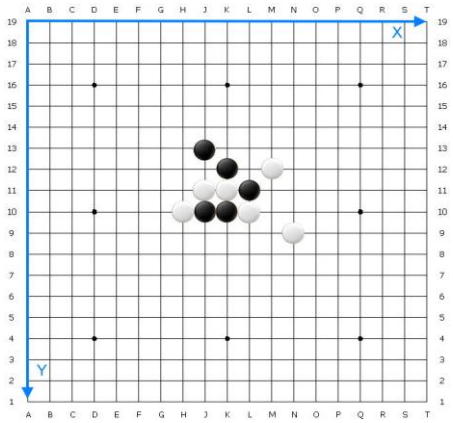

图 4.2 某棋局状态

Fig. 4.2 State of chess

表4.1六子棋棋子编码

Table4.1 chess code of connect6

棋子	黑棋	白棋	空位	界外
编码	0	1	2	-1

$$S = \begin{pmatrix}
2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2\\
2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2\\
2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2\\
2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2\\
2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2\\
2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2\\
2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2\\
2&2&2&2&2&2&2&2&0&2&2&2&2&2&2&2&2&2\\
2&2&2&2&2&2&2&2&0&2&1&2&2&2&2&2&2&2\\
2&2&2&2&2&2&2&1&1&0&2&2&2&2&2&2&2&2\\
2&2&2&2&2&2&2&1&0&0&1&2&2&2&2&2&2&2\\
2&2&2&2&2&2&2&2&2&2&2&1&2&2&2&2&2&2\\
2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2\\
2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2\\
2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2\\
2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2\\
2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2\\
2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2&2
\end{pmatrix} \tag{4.2}$$

具体实现上，本系统通过定义一个类，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坐标位置以及是何种棋用于转换后写入棋局状态矩阵 S 对应位置（也即 chessBoardStatus）：

```
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
    }
```

4.3.2 棋子状态数组描述

如图 4.3 所示，以落子点（图示黑子）为中心，横、竖、左斜、右斜 4 各方向左右各扫描 6 位得到 4 条“直线”对应的 4 组 13 位的棋子状态数组。在把这 4 个数组保存在一个 4×13 的的矩阵中得到该时刻的棋子状态矩阵 S^M，如式 4.2 所示。矩阵第一行到第四行分别对应搜索方位的，横、竖、左斜，右斜 4 个方向的数组。

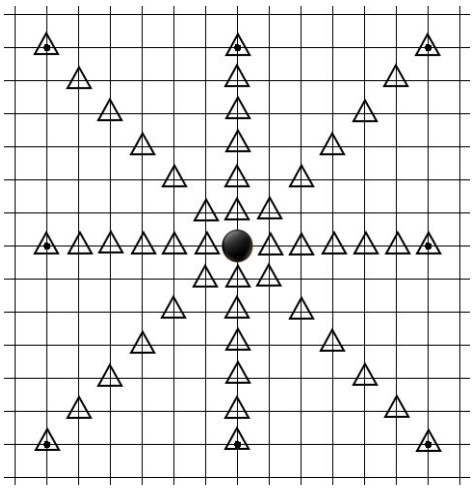

图 4.3　4 方向获取棋子落子位周围棋子分布状态

Fig 4.3 Get chess distributing condition around the new move in 4 directions

搜索方法为：首先获得当前落子位的坐标(x,y)，然后通过坐标变换到矩阵坐标（a,b）找到棋局状态数组 S 对应的表示该位置的数据 S[a][b]，以这个数据为中心，获得相应 4 个方向的棋子信息，形成 4 个 13 位数组 $S_{x,y}^{M}[i][]$ (i=0 to 3)。获取数据公式如下：

$$S_{x,y}^{M}[i][j]=\begin{cases}S[a-t][b], & i=1\\ S[a][b-t], & i=2\\ S[a-t][b-t], & i=3\\ S[a+t][b-t], & i=4\end{cases}(t=-6\,to\,6\ 取整数) \tag{4.3}$$

4.3.3 六子棋棋型及其数字化描述

由于 6 子棋除了黑白没有兵种之分，所以没有兵种参数，本课题组通过分析研究六子棋和其他连线棋的特点，总结出棋型的描述方式，使得各棋型就相当于象棋中的各类兵种。现给出六子棋常见棋型的定义和示例如下：

〖六连〗：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。

〖长连〗：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。

〖活五〗：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。示例：如图 4.3、图 4.4 所示。

〖眠五〗：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。图 5.3

〖死五〗：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。如图 4.5

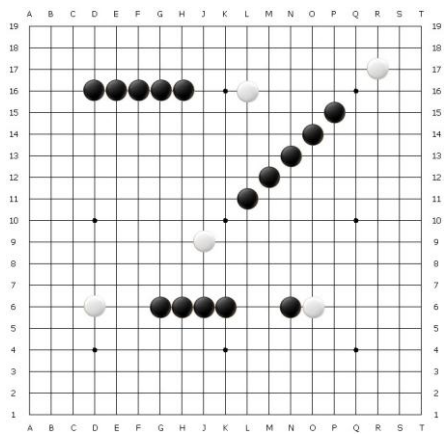

图 4.3 活五

Fig.4.3 Active five

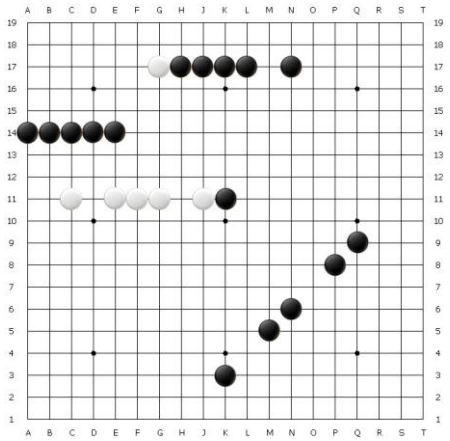

图 4.4 活五

Fig.4.4 Active five

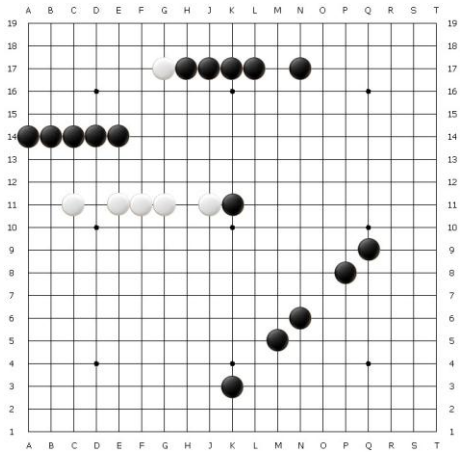

图 4.5 眠五

Fig.4.5 Sleep five

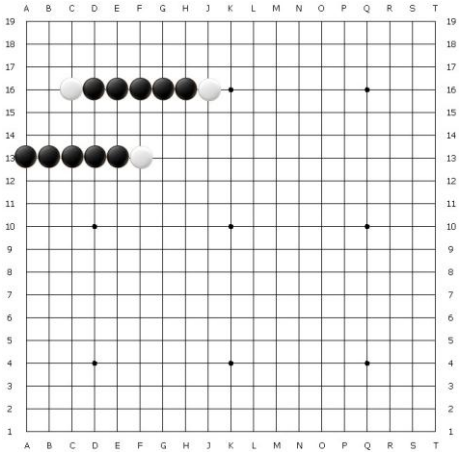

图 4.6 死五

Fig.4.6 Dead five

〖活四〗：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。

〖眠四〗：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。

〖死四〗：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。

〖活三〗：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。

〖朦胧三〗：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。

〖眠三〗：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。

〖死三〗：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。

〖活二〗：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。

〖眠二〗：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。

〖死二〗：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。

4.3.4 前期棋型数组表示的错误

只有准确的棋型，信息的提取才能准确，这事后续算法搜索和判断的基础。但本课题组前期在棋型的数字化表达上存在有很大的错误。

前期错误的实现方法是：把棋型定义为固定的 13 位的数组保存，以图 4.7 中所示活四为例，图中 a、b，c 都是活四，则三种活四的 13 位代码分别为(a)对应的 [1,2,2,2,0,0,0,0,2,2,2,2,2] 、 (b) 对 应 的 [2,1,2,2,0,0,0,0,2,2,2,2,2] 以 及 (c) 对 应 的 [-1,-1,2,2,0,0,0,0,2,2,2,1,2]，然后把所有这些活四的情况对应的数组都保存在棋型表中，试图以穷举的方式找完 13 位中所有符合活四定义的棋子分布来实现状态的不遗漏。

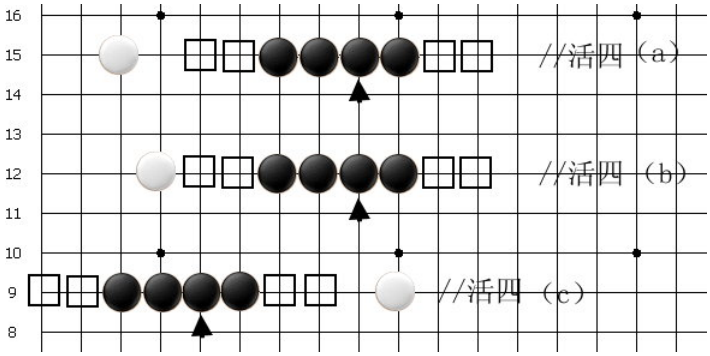

图 4.7 活四的几种状态

Fig. 4.7 Examples of active four

这种方法实现以后有两个主要缺点：第一，不可能穷举完所有状态，所以总是判断棋型时出现遗漏而导致搜索失败。图 4.7 中(a)所示的“活四”为例，只是一个白字的位置在 13 位范围内的变化就有 5 种，要是两个白字就有 20 种，三个白字就有 60 种，而且落子点位置不同数组又会不一样、有己方的棋子时又有变化等等各种变化，因此仅仅一个“活四”就按以前的描述方法为了全部匹配则要演变出上万数组。这还只是开局阶段，要是到了中局以及残局，棋子分布千变万化，人也无法预料到所有局面，所以穷尽完全是不现实的。这还只是活四的，眠五，朦胧三可能更多，也体现不出设计的智能。第二，随着保存棋型的表的规模不断扩大，尽管采用了哈希表的方式，但是速度仍然随着表的增加而降低，而且最重要的就是不准确的信息提取在怎么快也是不成功的。

这个方法明显是不可行的，实践也证明这种方法在实际操作中确实有错误，后期尽管把数组补充到两千多种还总是遗漏状态。

另一个不合理的地方在后面评估值的优化部分才会体现出来：由于我们采用的是利用遗传算法来解决评估值的调整问题，即先人为给棋型一个评估值，然后把棋型对应为遗传算法中的染色体，通过遗传算法和锦标赛方法离线计算出合适的评估值。但是对以遗传算法，几千参数的优化几乎是不现实的。

4.3.4 改进的棋型表示方法－特征码表示

为此，本文提出一种新的采用特征码的方法来表示棋型。首先给出一个本文提出的“无关位”定义。

无关位定义：例如图 4.8 中，黑箭头所示为落子位，根据定义，只要方框所示位置无子，都是活四，所以方框以外的位置都是“无关位”。

根据六子棋棋型的定义，各棋型都有其自己的特征，这也是我们特征码总结提取的基础。仍以“活四”为例，说明如何总结出其特点并定义一个特征数组的方法

和流程。如图 4.8，根据“活四”的定义，我们知道无论落子点在什么位置，最终“活四”的特征是：一行（直线）中，有 4 个同色的子紧密连续地连在一起，两端必须各有两个子位为空（图中方框所示），图中三角所示位和三角后面箭头所示方位的子位为无关位，子任意，即空，白，黑及其组合均可。为此活四的特征（以黑棋为例）用数字表达即为[2,2,0,0,0,0,2,2]，只需要一个 8 位的数组或者字符串就能描述“活四”的特点。

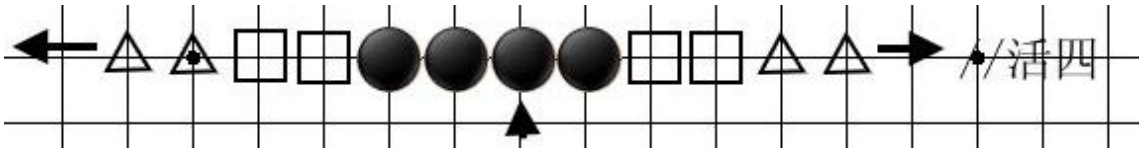

图 4.8 活四特征示意图

Fig.4.8 Description of active four characteristic code

如图 5.9 再以“眠五”其中的一个棋型为例（活四只有一个棋型，眠五对应很多棋型）给出该棋型的特征码：[1,0,0,0,0,2,0]。

图 4.9 眠五某一棋型示意图

Fig. 4.9 Character of one example of sleep five

可以看出，这种方法的好处：特征就是特征，固定，不受各无关子和落子点位置的影响，如图 4.8 中，无论落子点在那，特征始终是上文描述的那些特征。而且某一个棋型的一个特征棋型只有一个固定的特征码。该方法舍弃了 13 位数组表示，无关位补空的方法，通过这种方法最终前文叙述的棋型最终就只有两百多种，而且每个棋型数组表示也得到简化。表 4.2 为保存各棋型特征码（以黑棋为例）的表的一个片段。

评估函数中仍然采用以前的 4 方向各 13 位的方法得到实时的棋子分布，然后在每个方向的 13 位数组中，分别匹配棋型的特征子数组，即比如查找是否有活四，则只需在数组或字符串中查找是否有[2,2,0,0,0,0,2,2]子数组或子字符串。若有则存在活四。这种方法每个方向都要查 200 多种棋型数组或字符串来匹配，表面上看起来很耗实践，但是以现在的计算机能力，实际在计算机处理的时候是几百个字符串的匹配搜索是十分快的，而且有很多成熟的字符串查找的高效算法可以直接应用，使得搜索更快。而且，有相关棋类知识的人员都可以发现，理论上查找一个方向需要匹配两百多次，但是实际在搜索中，都是假设对方会走最好的，我方

也走最好的，所以在中局的时候大多数是搜索到“活四”程度左右的时候大多都已经找到匹配，不用进行后即搜索。以搜到“活四”为例，只需搜索 30 多个子串，后续不用搜索。因此，这种方法的实际效率可以接受。

表 4.2 棋型特征码表

Table4.2 table of character code of chess type

棋型名称	特征码
连六	000000
活五（1）	2000002
活五（2）	22000020
活五（3）	220000220
…	…
眠五（1）	1000002
眠五（2）	1000020
眠五（3）	12000020
眠五（4）	10000220
…	…
活四	22000022
眠四（1）	1000022
…	…

必须需要注意也是应用这种方法很关键的一点就是：搜索必须从高级棋型往低级棋型的顺序搜。即：先连六，然后活五，然后眠五，然后死五，然后活四…，直到最后都没有搜到就判断没有匹配。

4.4 六子棋计算机博弈问题描述－博弈树

六字棋计算机博弈的描述自然也要用到博弈树，图 4.10 即为六子棋的部分博弈树。 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。

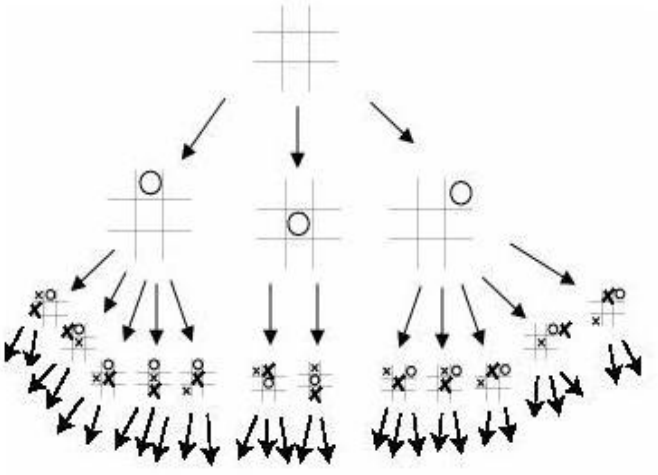

图 4.10 六子棋的博弈树

Fig. 4.10 Game-tree of Connect6

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。

4.5 优化的搜索算法－基于启发式信息的搜索

6 子棋的搜索就是围绕以上的博弈树展开的，目的是如何搜索最少的节点却能获得最大的效果。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依然采用了 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来优化搜索。

4.5.1 α-β 剪枝搜索算法的实现

在搜索算法的选择上，本文通过对比和分析，决定采用最基础的 α-β 剪枝算法配合启发式信息优化出一个适合六子棋的改进算法，α-β 算法几乎是现在所有主流

搜索算法的基础，也为我们以后的优化提供了无限可能。但是经典的博弈搜索算法实现的基础几乎都是一次走一步的规则下优化产生的，而 6 子棋是一步两层，所以势必对所有现存的算法都作出适当的改进来优化成适合 6 子棋博弈树的算法。

前期的系统设计只是采用了递归的方法对两个 13 位数组一一匹配的方式，效果不是很好。本文在决定采用 α-β 算法之后仍然面临两个选择，一个是两步递归一层剪枝，还是每步递归一层剪枝每轮递归两次。这两个的效果是明显不同的，后者是把一轮的两步分开考虑，势必会丢掉不少两个着法之间的关联的信息量，所以放弃。α-β 剪枝的模型见本文第 3 章图 3.4、3.5，此处只说明如何将 α-β 剪枝搜索算法实际引入到六子棋计算机博弈系统的搜索引擎模块，实现方法如下：

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。

α-β 剪枝伪代码如下：

```
Int alphabeta(chessboardstatus,depth,alpha,beta)
{ if(game over)
        Return;
  If(n==0)
  Return evaluation;                    //0层，叶子节点，返回评估值。
  If(MIN NODE)                          //如果是极小节点，beta剪枝
  {   For(m1=possiblemove(chessboardstatus)) //第一步所有可能走法
      { make move m1;
        For(m2=possiblemove(chessboardstatus)) //第二步所有可能走法
        {Make move m2;
       Stepvalue = alphabeta(chessboardstatus,depth-1,alpha,beta); //递归
        unmake move m2;
        }                                   //还原
        unmake move m1;
        if (stepvalue<beta)
        {nbeta = stepvalue;
        if (alpha>=beta)
        rerurn alpha;
        }
      }
      Return beta;
  }
  Else
  {……}                                  //极大节点，α剪枝类似beta剪枝，略。
}
```

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。

4.5.2 优化的 α-β 剪枝搜索算法－启发式搜索

在六子棋计算机博弈系统的搜索引擎模块中，已经成功应用了 α-β 剪枝搜索算法。但综观现有棋类的计算机博弈中，完全靠单一的原始搜索算法来搜索很难得到理想的搜索结果，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。而且一般的棋类都是双方每轮各走一步，博弈树扩展时按一轮一层设定搜索 3 层深度，步数实际也是 3 层。但六子棋除由于其规则的特殊：除了了第一步外双方每轮都是各走 2 步，其步数为 6 步，实际搜索深度是 6 层（一步一层）。所以，在六子棋程序设计中，设计各种好的启发式方法使树的规模减小显得尤为重要。但是也象前文所述，优化的方法和信息十分丰富，本文受时间限制，也只是选择了其中的几种进行优化测试，但是都取得了不错的效果。

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，这些信息主要来源于棋类自身的特点，比如六子棋的战略，以及借鉴的前文提到的迭代深化和单步延伸的思想，开局库以及 VCF 的应用。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信息分支条件相匹配，则按照相关信息下的搜索法则来进行后续的搜索，如果不匹配，则继续传统的搜索过程。

① 启发式信息－单步延伸和威胁判断

本文采用的启发式方法是结合前文介绍的单步延伸、迭代深化的思想和威胁判断策略（威胁定义见前文）。显然，对于 6 子棋，对敌方造成三个或以上的威胁就赢，因此，赢的策略就是在阻挡所有对方的威胁同时，产生三个或以上的威胁。通常一个三威胁由一个双威胁和一个单威胁组成，而判断双威胁就可转化为前文所述的用棋型特征码判断有无活四或活五，相应的单威胁也可以找到很多相应的棋型对应，此处不一一列举。

威胁判断：把威胁判断加入博弈平台实现模型的 α-β 递归搜索函数的递归部分前面，从而在博弈树递归扩展之前，先做各种威胁判断：包括敌我双方，先判断己方目前有无形成双威胁，若有，直接连六获胜（对于常规扩展时连六状态在后面叶子节点才出现的博弈树，此举可以减少搜索很多分支），若没有，则继续判断对方有无三威胁，若有则认输，没必要搜索，若没有三威胁，则继续判断对手有没有双威胁，若有则只需要以该双威胁为根节点从新扩展堵住该双威胁的招法，不用继续扩展之前一层后续所有招法，因为面对双威胁己方肯定要防守，否则就要输，所以其他分支搜索无意义，这就可以减少搜索很多无用分支。最后是判断对方的单威胁，若有，先阻止，然后在启动扩展，分析第二步的走法。以上这些处理方法可以说是对应情况的固定对策，所以不用搜索，直接在博弈树扩展前就

把方法单独提出“固化”处理方法。若是上面的威胁暂时在该层都没有搜到，就要递归来进行比较复杂的潜在威胁招法判断，比如活三、死三、活二，这些棋型每次下两颗子就有机会形成真正的迫着。对于这些状况本文没有单独提出模块，而是在实现过程中通过评估函数结合适当的评估值来判断。图 4.11 为一个简单的三－双威胁的策略流程图。

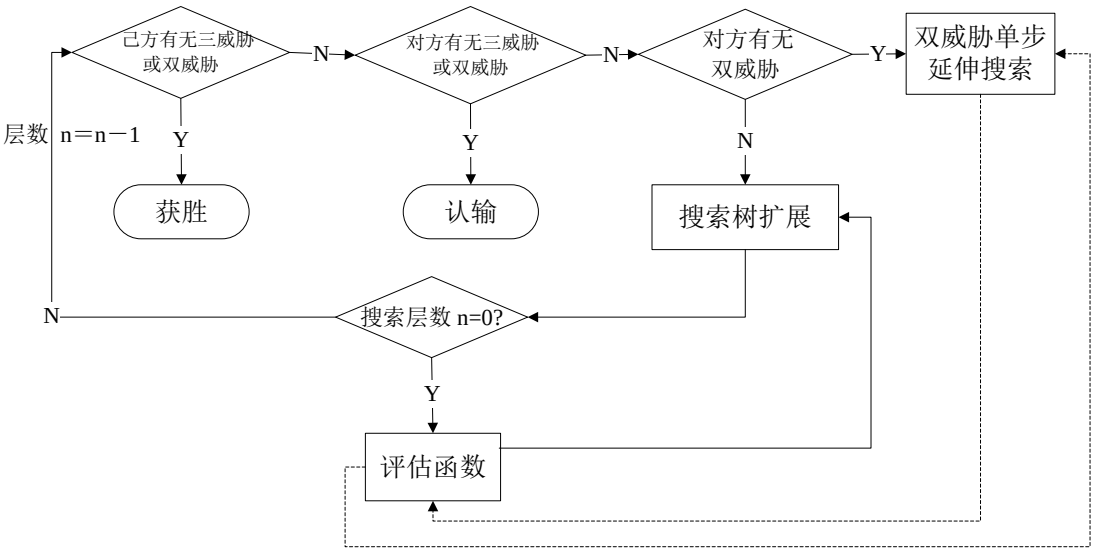

图 4.11 加入威胁判断和单步延伸的改进搜索流程图

Fig. 4.11 The improved process of researching based on threats and Singular Extension

这个流程图对原图的改进在于在每次搜索扩展的前端加入一些启发式的触发搜索条件，只有这些条件不被触发的时候才会进行后续的传统的扩展搜索，可以有效的减少搜索的数量和时间。

通过实际对弈证明，这种方法有效的缩短了搜索时间。这个策略中采用的威胁分支，但是这种分支的条件可以是很多，比如一个双威胁和一个眠棋的威胁组合，也是值得分支的单独搜索的，本文只是以最好理解的威胁判断来说明，但是也可以看出，对于那种状态值得分支十分依赖于设计者的棋类知识，需要不断的优化才能最终达到最好的状态。在本文写作过程中，本程序只是加入了所有典型的三威胁和四威胁分支。

② 启发式信息－对于搜索范围的优化

除了算法本身的优化外，博弈树的优化同样可以达到提高搜索效率的目的。本文在实现上结合 6 子棋的特点，反应用前文所述的脱离战场策略分析得到：某一时刻发生“交战”的范围只有一个固定区域，而且比较集中，其落子的影响也只是一个有限的范围，别的区域或者已经是“死”，或者还是空白（棋盘上没子），因此

本文在的搜索的时候完全没必要搜索那些空白区域或者死域。

为此，本文根据 6 子棋所有子都分布比较集中这个特点，提出一种方法：通过人为限定可行招法搜索范围的方法在几乎不损失准确度的情况下达到大幅度地减少搜索量，提升搜索时间。方法如图 4.4 所示，在图中棋子集中区域，以上、下、左、右 4 个方向最边界棋子的坐标作为边界勾画出一个矩形区域，图 4.4 中黑色实线框就是当前棋盘内地棋子地分布矩形区域。然后以这个矩形为基础，每个边都扩大两个棋子位，“画出”一个新的虚线所示的矩形区域。以这个虚线区域代替整个棋盘作为可行落子点的搜索范围，避免了全盘搜索。实际效果证明，相对准确度影响很小。但是搜索时间有显著的提高，表 4.2 为限制范围前后的时间对比，可以看出，限制了搜索范围后搜三层的时间也比全盘搜索搜一层的时间要快很多。

本文提出的这种方法利用了脱离战场法则的思想，由于落子位置距离棋子分布区域太远子力就会十分弱小，属于完全没有意义的棋，如上图所示，如果在四个角的位置落子，完全对中心区域的战局没有丝毫影响（除非计算到后续几十层甚至直接扩展到棋局结束状态，而这两个方法对于计算机计算几乎都是不可能的，所以在所能扩展的计算范围内，改位置对战局没有影响）。之所以选择扩展两个棋子位，是根据连 6 赢的特点，如果在三个棋子位以外落子的话，无论如何也不会产生很大的威胁，因为中间空了 3 个子位，而一次只能下两子，树无法一手就赢的，至多只能产生眠棋的效果。但需要注意的是只是，搜索范围的限定只是在搜索所有可行落子点时使用，在判断棋型状态时使用 13 位数组不受范围限制，依然以落子点为中心左右各扩展 6 位来判断含有什么棋型。

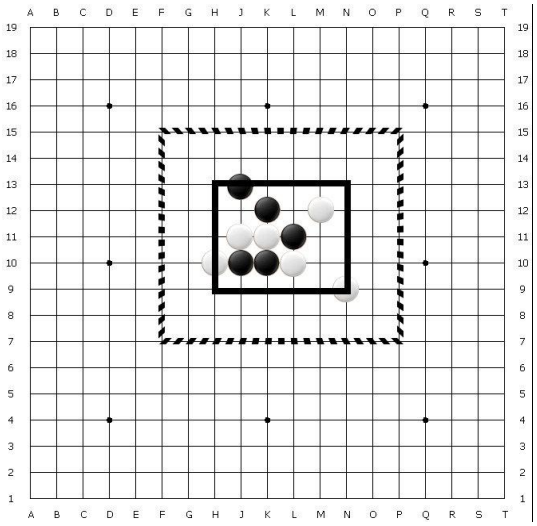

图 4.12 搜索范围图

Fig. 4.12 The area of searching

	全盘搜索	限制搜索
搜索深度	1	3
走两步所用时间	平均 20 秒	平均 2 秒

表 4.3 搜索时间对比

Table 4.3 The comparation of time used

实现方法如下：

首先在系统中定义 4 个边界指标值：leftegde、rightedge、TopEdge、BottomEdge，初始化为一个用来保存和实时更新矩形范围的 4 个边的坐标，然后没走一步棋，都要和这个 4 个边界当前保存值对比，如果某一方向超出边界，则该方向的边界就是这个新落子的边界，在扩展搜索时一这个新边界外括两层展开博弈树搜索。

```
private void CalEdge(int x, int y)   //计算边界的函数
{
    MainboardStatus.LineUsedStatus1[x] = true;
    MainboardStatus.ColUsedStatus1[y] = true;
    if (MainboardStatus.LeftEdge > x)
        MainboardStatus.LeftEdge = (byte)x;
    if (MainboardStatus.RightEdge < x)
        MainboardStatus.RightEdge = (byte)x;
    if (MainboardStatus.TopEdge > y)
        MainboardStatus.TopEdge = (byte)y;
    if (MainboardStatus.BottomEdge < y)
        MainboardStatus.BottomEdge = (byte)y;

}
```

之后就需要即选需要扩展的范围，其实现方法很简单，比如判断左边界，首先判断是不是扩展后会不会超出边界，若超过则扩展边界设定为 0 或者 18，否则加 2 或者减 2 获得一个新的搜索扩展节点边界。以左边界为例说明实现方法如下式：

$$LeftStart = (LeftEdge < 4) \ ? \ 0 : (LeftEdge - 4) \tag{4.4}$$

其他方向均类似。

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。其实各个部分都有启发式信息的应用。比如评估函数对于如何评估就加入了棋类的特有知识。以后在每个部分分别叙述。

4.6 本章小结

本章中简述了本文六子棋计算机博弈系统平台实现的方法，平台完成后主要针对搜索流程和博弈树的扩展规模进行了优化。搜索上采取了 α-β 剪枝搜索算法，以及在系统的搜索流程上加入了新的启发式分支模块，用于简化搜索，使得搜索的效率和准确性都得到了进一步的提高；之后又反利用脱离战场策略删减掉很多冗余的节点，大大提升了搜索的时间。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解决节点估值的问

题。

并附上系统最终的实现图，该图为采用所有改进后的最新的实现程序图，为正在对弈中的程序截图 4.5。

图 4.5 巴将军对弈过程截图

Fig.4.5 chessing snapshot of general-bar connect6

5 评估函数优化及遗传算法优化评估值

5.1 引言

由于改进了棋型的数字表示方法，使得局面信息的提取更准确，从而对前期系统带来了很多改变，如评估函数的优化，考虑进去了位置信息；摒弃了棋型演化的判断，而是把可能的棋型演化直接体现在特定棋型的棋型值上。最后，新的棋型特征码方法的还使得遗传算法优化的评估值参数变得实际可行。

5.2 局面评估方法

局面评估的基础方法如图 4.3 所示，如前文所述获得落子位周围 4 个方向的棋子状态数组 $S_{x,y}^{M}[i][]$，得到棋子状态矩阵，然后调用表 4.2 中的棋型特征码与之前的 4 个棋子状态数组依次进行数组的匹配搜索，判断是否含有某种棋型（即数组中是否含有棋型特征码数组串），有则再调用权值表（一个棋型对应一个评估值）获得该方向上该棋型的评估值，如果某一方向有多个棋型，则依次累加。

定义 value 为总的评估值，Stepvalue_i 分别为水平、垂直、左斜，右斜 4 个方向各自的评估值。

最基础的评估值计算方法为：

$$\text{Value}=\sum_{i=1}^{n} stepvalue_i \ ,\ n=4 \tag{5.1}$$

基础实现是按上式实现，但是可以看出，式 5.1 中没有体现出连子棋的特点，就是各方向棋型之间的相互影响的配合关系，所以简单的相加不足以准确的表达该局面。故提出后文将要描述的加入位置信息的评估值计算方法。

5.3 评估值和评估函数的改进

局面的评估必须要有棋型的数字化表示，函数才能判断对应的是什么棋型。然后必须再给各棋型一个的数字评估值，评估函数才能对局面有一个量化的评价。

以下是采用这个方法后评估值和评估函数作出的一些改变。

首先，由于取消了前期根据状态演变来给棋型设定评估值的方法，所以，在棋型的评估值设定上必须对应每种棋型把状态变化的因素柔和到固定的棋型评估值中。

其次，而且评估函数的棋型匹配中，也不是每个方向只要搜到一个棋型就放弃不搜。而是搜到一个棋型后，保存断点，然后继续从断点重新搜索。

最后，在评估函数中加入了一点形状考虑因素，在研究交大 6 号的时候发现“冲

三角”活四是一种很好的取胜策略，所以在搜索中要是预估到要是有两个方向能形成三角顶角，则适当的再给一个加权，使其比简单的相加得到的综合评估值要高。

$$stepvalue=\begin{cases}a*(step_H+step_v)+b*(step_{LS}+step_{RS})\\ a*(step_H+step_{LS})+b*(step_v+step_{RS})\\ a*(step_H+step_{RS})+b*(step_v+step_{LS})\end{cases} \tag{5.2}$$

其中 $step_H$ 、 $step_v$ 、 $step_{LS}$ 、 $step_{RS}$ 分别为图 5.6 中所示的水平、垂直、左斜，右斜 4 个方向的评估值。通过对三角形成的方位不同，a 和 b 加权系数对应的加权对象组合不同，而且 a,b 对应不同组合，取值也不一样(a,b 取值范围为 0，或者大于 1)。比如式 5.1 中的第一行就表示水平扫描状态和竖直扫描状态匹配或左斜和右斜匹配的可能。b 为 0 就代表只有一种可能，非 0 代表两种组合同时存在。

本方法的应用使得评估进一步准确，所以证明方法和方向是合理的，今后随着设计者棋类知识的丰富会加入更多的类似策略，所以在设计搜索的时候都是以模块函数的形式设计的，使得以后的加入就像“搭积木”一样，只要在适当的流程环节上加入一个分支判断模块即可。

5.4 遗传算法的实际操作

5.4.1 奠定应用遗传算法的基础

在前期的系统设计中，我们曾提出用遗传算法来优化我们人为给出的棋型评估值，为了应用遗传算法，我们把每一个存储键值（棋形数组）的哈希表做为一个个体，哈希表中的每个棋型的评估值对应为该个体的一个染色体，从而提出了应用遗传算法的方法。但是在实际操作中，这几乎是不可能实现的，按前期系统的设计，每个个体（也就是哈希表）中的棋型有上千种，自然评估值也就对应有上千种，而且由于棋型信息表示的失误，随着人为的不断加进遗漏的棋型状态，这个表（个体）是不断扩大的，遗传算法对于少参数的优化时间效果比比较好，对于几千个参数优化，实际中用遗传算法操作尽管可以实现，但是要达到预期优化效果所需的时间几乎不可计算，显然要想短期内达到预期的效果是不现实的，所以实际可用性不大。

但本文后期引入了新的棋型特征来表示棋型后，首先棋型缩小到只有两百多，而且这两百多在遗传算法的操作过程中不是全部参与，因为其把活三变活四之类的状态演化的因素直接体现到自己的数字评估值中了，比如对于眠五，有六种棋型，但是把棋型变化考虑到评估值中了，所以在用遗传算法的时候，只需要从六种“眠五”的棋型中随便提出一个做为眠五的代表就可以了。因此，总的参数需要优

化的，除了朦胧三的状态外，每个棋型只有一个（连六和长连不优化，必定是无穷大）。这样就只有 11 个，方法仍然采用系统之前设计好的遗传算法优化程序，编码方式也不变，只是保存一张存有单独的 11 个棋型的表参与优化。大大缩小了计算量，使得遗传算法的应用有了很大的可实现性，但由于遗传算法的计算很需要时间，为此在论文的写作当中，算法还在计算中。

5.4.2 遗传算法优化棋型评估值的操作过程和方法

① 锦标赛选择

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，本文采用了锦标赛选择法和精英选择策略[42]。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。锦标赛模式的缺点在于速度过慢。为了加快速度，本文对于标准锦标赛做了改进。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。分别在各组内进行锦标赛训练，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。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，以包含 20 个个体的种群为例，在保证优胜劣汰原则的基础上，可以节省约 80% 的时间[40]。

② 均匀交叉

交叉方法很多，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，参数之间的间隔点作为潜在的交叉点。均匀交叉根据交叉率 Pc 随机地产生与参数个数等长的 0 - 1 掩码，掩码中的片断表明了哪个父个体向子个体提供变量值。通过这个掩码和选择的父个体一起确定子个体。

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。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。

5.4.3 改进的遗传算法

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。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。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，本文引进了一个改进的遗传算法——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：

$$pc = \begin{cases} pc1 - \dfrac{(pc1 - pc2)(f' - f_{avg})}{f_{\max} - f_{avg}}, & f' \geq f_{avg} \\ pc1, & f' < f_{avg} \end{cases} \tag{5.3}$$

$$pm = \begin{cases} pm1 - \dfrac{(pm1 - pm2)(f_{\max} - f)}{f_{\max} - f_{avg}}, & f' \geq f_{\max} \\ pm1, & f' < f_{\max} \end{cases} \tag{5.4}$$

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。

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。

5.4 实际操作结果对比分析

本文的实验效果证明主要有两个方式，一个是通过和世界最优秀的程序对弈做比较分析，一个是和设计者或者人进行对弈做比较分析。下图为实际操作程序界面。

由于本方法十分耗费时间，截至本论文截稿日期，以上实际操作工作还远远没有达到应该操作的代数，只操作了大约 20 代左右，但是效果已经微显，优化后的参数甚至已经有战胜世界 6 子棋冠军交大 6 号的记录，尽管次数只有几次。但相比以前几乎坚持不了 30 手就输已经有了很大的提升而且最主要的是证明方法的有效性和想预期的好方向发展，后期的继续操作是有价值的。

在和人对弈方面，由于时间原因没有做过具体的数据统计，但是已经至少能坚持到 40 手不露败迹。下表为一张优化后的评估值表的片段。

表 5.1 评估值表片段

Table 5.1 part of evaluating table

	连六或者长连	活五	眠五	死五
评估值	100,000	55,579	38,201	0

5.5 本章小结

本章主要阐述了棋型表达的新方法，以这个方法为基础对系统各部分进行了优化后，系统整体有了一个飞跃的提升。在搜索中，搜索不会再遗落状态做到信息提取准确。在评估函数中，由于搜索方法的精确使得可以加入形状的策略，并且在加入该策略后，效果确实略好于没有假的效果，但是由于是人为设定加权系数，所以还不是很准确。最后，使前期提出的离线遗传算法优化评估值的方法向实际操作大大的迈出了一步，由于时间和精力原因，尽管只遗传了少量后代作为测试应用到程序实际对弈中，实际对弈中的第二步计算机自己利用优化后的方法和评估值，选择的走法恰好是 6 子棋中最常用功防兼备的山水局。由此可以验证效果还是有的，还是向好的方向发展，已经可以直观的感觉出棋力的提升。

6 系统功能的完善

6.1 引言

在本论文以前的前期工作中，系统只是实现了简单的对战功能，而且还只能计算机为黑先，本论文的实践过程中不断丰富了系统的功能，加入了棋谱保存、悔棋、人机任意对战等等。

6.2 程序的完善

图 6.1 新的程序最终实现结果

Fig. 6.1 New results of connect6

6.2.1 悔棋的实现

利用一个单独的数组和 C#的 imagebox 属性实现多步悔棋。实现方法如下：

定义两个整型变量：stepcount（用于保存走棋总步数），ChessBackCnt（保存悔棋步数）。

在悔棋时，首先要判断是否在走棋当中，如果是则不能悔棋，只有走完一步棋后才能悔棋，这个通过一个标志位 chessing 来实现。当可以悔棋时，就要判断当前的走棋状态，因为 6 子棋是一次两步，所以还要判断当前是走一步后悔棋还

是 2 步后悔棋。

通过 picturebox 的属性确定要删除的棋子的位置，然后在利用 chessboard.control 的 remove 控件属性（我们的棋子是以 picturebox 控件的形式落子的）移除这个要悔棋的棋子，最后必须更新棋盘状态矩阵 S。以下定义实现函数：

```
private    PictureBox[]     AllStepsMap; //  全局变量，用于保存历史棋谱
private void ERROR_PRESS_Click(object sender, EventArgs e)
{ if (chessing == false)   //条件是：  还没有开始，或者胜负已定
    {
      MessageBox.Show("棋局还没有启动或已结束!！ ", "不能悔棋");
          return;
     }
    this.chessboard.Enabled = false;
    chessing = false;
    timer.Stop();
    PictureBox pb;
   if (ChessMovedCnt == 0)
      if ((side == 0) == !BlackIsComputer)         //side = = 0，该黑方走了
        {   ////////////////////移除要悔的棋子////////////////////////////////
            pb = AllStepsMap[stepsCount - 1];
            chessboard.Controls.Remove(pb);
            stepsCount--;
           ////////////////////更新棋盘状态矩阵/////////////
     chessBoardStatus[(pb.Location.X + 10) / 24 - 1][(pb.Location.Y + 10) / 24 -
1].player = 2;
      chessBoardStatus[(pb.Location.X + 10) / 24 - 1][(pb.Location.Y + 10) / 24 -
1].used = false;
           ////////////////在棋谱保存数据框中移除相应的棋谱记录/////////////////
         historySteps.Items.RemoveAt(historySteps.Items.Count;
  SelectedChessman1.Location=new Point(AllStepsMap[stepsCount-2].Location.X+4,
          AllStepsMap[stepsCount - 2].Location.Y + 4);
          SelectedChessman2.Location = new Point(AllStepsMap[stepsCount -
          1].Location.X + 4, AllStepsMap[stepsCount - 1].Location.Y + 4);

            ……..                    //后续原理重复以上步骤，略
}
```

6.2.2 棋谱保存

利用 C#的文档写入函数把棋谱保存为文档文本，可以便于后期分析总结。实现方法如下：

```
      private void SAVE_GAME_Click(object sender, EventArgs e)
       {
          this.saveFileDialog1.RestoreDirectory = true;
          this.saveFileDialog1.CreatePrompt = true;
          this.saveFileDialog1.InitialDirectory = ".\\saved_chessboard";
          DialogResult ppp = this.saveFileDialog1.ShowDialog(); //获取文件句柄
```

```
                if (ppp == DialogResult.OK || ppp == DialogResult.Yes)
                { ////////////////////利用 stream 流来写文件//////////////////////////////
    System.IO.StreamWriter myfile = new System.IO.StreamWriter
(this.saveFileDialog1.FileName, false, Encoding.Unicode);
                    if (BlackIsComputer == true)
                        myfile.WriteLine("The Black side is Computer!");
                    else
                        myfile.WriteLine("The White side is Computer!");
                    for (int k = 0; k < historySteps.Items.Count; k++)

myfile.WriteLine(historySteps.GetItemText(this.historySteps.Items[k]));
            ////////////////////利用 historystep 控件的 item 属性//////////////////////////////
                    myfile.Close();
                   MessageBox.Show("棋谱文件保存成功！", "棋谱文件保存成功！");
                }
                else
                    MessageBox.Show("棋谱保存失败");
            }
```

棋谱保存的文档截图如图 6.2：

图 6.2 棋谱保存演示

Fig. 6.2 demo of chess-saving

6.2.3 其他优化

程序的计算机搜索模块没有直接调用搜索函数，而是通过一个线程启动模块，这样的好处是为以后实现计算机相互对战时的多线程的运行提供基础，利于实现自动机机对弈以加快速度。但当前没有实现机机对弈的多线程运行，只是为这个

思想而预留了一个方式。

实现代码如：Thread thread = new Thread(new ThreadStart(computerThink));

其中 computerThink 就是本 6 子棋博弈系统中计算机搜索函数模块。而 new thread 就是启动新的线程运行参数中我们设定的程序或者模块。

6.3 本章小结

本章对系统实现了功能的扩展。现在系统具备了悔棋，棋谱显示，棋谱保存等功能，使得平时的调试和研究更方便，程序也更完善。

7 结论与展望

7.1 结论、比赛结果及系统评价

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，而且由于去规则简单确具有很高的复杂度，研究价值比较高，所以逐渐获得学术界的关注，继 ICGA 和台湾把六子棋作为计算机博弈竞赛项目之后，中国也于 2007 第二届博弈竞标赛中设立了六子棋项目。

比赛结果及分析：

本文实现的系统也参加了 2007 第二届中国计算机博弈锦标赛，并且取得一定的成绩。先后手的区别对本棋很重要，比赛中，基本都是平局，一般先手的都会赢，后手的都会输，而本程序是唯一一个能无论先后手都能取得胜利的程序，但很可惜的是由于操作上的失误，在第二天调试程序阶段特征码表的边界数据被失误覆盖，导致了后续的程序判断出现错误，虽然仍然勉强参加后续的比赛，但效果只取得一平的战绩，通过分析棋谱发现所有输的局面都是在对弈到棋子分布扩展到边界时，由于之前边界数据被错误覆盖时才导致判断错误从而认输，而不是评估体系的问题。但是仍能从对弈上看出一些有用的信息：在没有涉及边界判断的最初阶段，棋力还是十分高的，基本是处于高威胁进攻状态，只是在程序出错部分判断棋型失误导致搜索失效。另外在程序出现错误之前完胜的对手都是在比赛后排名在前面的程序，也就是棋力都比较高的程序，说明我们的方法的有效和正确性。

本文主要工作是对各个模块的优化和新功能的扩展，使得系统的智能提高了很多。其最终实现结果如上一章图 6.1 所示，程序首先从功能上讲是可以完整高效地运作的，其次从智能上讲具有了一定的智能，棋力有了很大的提高（对弈取胜几率大大提高）。

本论文主要内容和结果如下：

① 前期和本组师兄一起完成平台，之后主要六子棋并对六子棋的棋型做了特征总结，并给出特征码描述方法，摒弃了状态演化的过程。实际对弈结果证明在优化后的比赛中再未出现状态遗漏的现象，从而使得信息的提取十分准确，为后续各部分的优化奠定了坚实的基础。

② 针对六子棋一次两步的特点实现了改进的 α-β 剪枝搜索算法，并加入了六子棋的启发式信息，结合 α-β 剪枝搜索算法构成一种适合六子棋的启发式搜索方法。

③ 借鉴了单步延伸的思想，优化了通用计算机博弈平台的结构，加入了策略

分支的判断搜索，提高了搜索效率。

④ 对博弈树的规模同样进行了有效的删减，通过脱离战场策略的反面应用，使得搜索树的规模在不影响准确度的情况下大大缩小。使得搜索更集中，也更快。

以上三点都是对于计算机博弈搜索相关的优化，三者相辅相成构成有个整体，从而在时间指标上有了显著的提升。通过前文的时间对比可以发现新算法在时间上的高效性。

⑤ 由于棋型描述的优化，使得遗传算法的所需要优化的参数缩小到可以接收的范围，只有 11 个，使得前期提出的遗传算法加锦标赛方法优化棋型评估值的方法具有了实际可操作性。但由于锦标赛算法优化工作需要一个相对有些长的时间，所以截至论文终稿，也只遗传了十几代左右，但是效果已经初显，相信随着优化的继续进行，一定能达到一个比较好的效果。

⑥ 最后，就是扩展了通用平台，加入了一些其他模块，如补充了棋谱保存、悔棋功能模块，使系统更完善。

通过以上所有方面的优化，系统整体的性能相对于优化前有了很大的提升，和传统的 VCF 实现的程序相比基本可以完胜，证明了时间指标的提高（也即时间缩短）并不是以损失准确度为代价的，或者说不是以损失可扩展范围内的准确度为代价。

在实际测试中和基于 α-β 剪枝算法的程序对弈，结果平局，但是是在对方每走两步需要将近 30 秒，而本系统只需要 2 秒左右的基础上，时间意味着可以搜的更深更广。将传统的时间调到和本系统相近的时候，本系统基本全胜。

对博弈树复杂度，由于采取了限制博弈树扩展，使得节点的扩展大大缩小，在优化前的节点数每一层基本是 19×19（后期随着空位减少而减少，但是以 30 手计算仍然保持在 15×15 左右）所以规模仍然十分庞大，而采用限制搜索策略后，博弈树规模随时变化但基本保持在 9×9 的范围。对于状态复杂度，因采用的 α-β 剪枝在剪枝统计上很难评估是因为加入太多的启发性信息，启发式的分支判断就会放弃很多节点而不进入搜索流程，所以只能从整个搜索模块的角度看状态空间复杂度。α-β 的有效性到底有多少取决于节点的扩展方法，在最优的情况下，对于一个结点为 N 的博奕树, 采用 α－β 剪枝进行搜索, 一般可以把搜索的时间效率提高到 o($\sqrt{N}$), 这意味着其搜索深度, 可以提高到未采用剪枝算法的 2 倍, 但是本文实际启发性信息每局都在随对手而变化放弃节点数目，所以只能大概预估至少减少了 40％（因为算法评估是固定的，所以可以根据时间和博弈树复杂度来推算状态空间复杂度），算法的评估有时要优于 O（ $\sqrt{N}$），但有时又低于 O（ $\sqrt{N}$）, 这都取决于启发式信息的效果。因为大多是 6 子棋的对弈中主要出现的棋型都是活四、眠五等棋型，这些棋型的出现总是会诱发启发式的信息判断，从而协助算法剪枝。

本文的创新之处有以下几点：

① 用特征码描述棋型。不受其他棋子和落子位以及位置的影响。

② 在搜索引擎和走法生成模块中，研究了脱离战场策略，人为限制搜索范围。在搜索中，把搜索流程模块化，并实现了基于策略的分支搜索。

③ 使遗传算法在六子棋中的应用由理论变的可行。

7.2 本系统目前存在的问题和不足

到目前为止，程序已经，而且具有一定的棋力。但是本系统还存在以下问题和不足：

① 由于竞标赛方法的特殊性，使得遗传算法对评估函数参数的优化效果的最终结果还不明确，只是目前看效果还可以接受。

② 搜索过程还有很多启发式信息可以加入，本文只是加入其中的部分而已，随着设计者对六子棋的研究的深入，会有更多的策略加入来提高搜索效果。

③ 两部当作一层的搜索方法还是比较粗糙，目前还没想到好的解决方法。

④ 通过和其他学校的交流，发现了一种新的方法－VCF，目前正在研究中，还没有具体应用进去。但是以后 可以考虑作为一种启发式信息加入的搜索过程中。

⑤本文写作过程中，正在尝试一种基于神经网络的评估函数，但是由于还在测试过程中,效果也不明确，所以还需要后续的工作。

⑥ 残局和开局还是没有加入。

总的来说，系统的智能高低很大部分取决于设计者的棋类知识，本文的很多工作都是把设计者的知识加入各个模块来提高棋力。

7.3 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：

① 为了验证遗传算法算法在 6 子棋上的优化效果，竞标赛算法的方法还得继续，且需要很长一段时间。

② 由于精力和资源的限制，还是没有加入残局和开局，所以后期的工作主要是完善这个部分来提高棋盘的中局和残局能力。

③ 搜索算法的优化可以说永无止境，一方面可以通过加入策略来优化，一方面可以把两步一手的搜索更精确化。

④ 本实验室已近开始尝试在 6 子棋平台引入神经网络做系统评估加权部分的优化，但是目前还在调试和测试，故文中没有体现。相信神经网络的实现对于系统的提高一定有很大的作用，尤其是权值的设置，将使得程序逐渐的智能化，脱

离对设计者一人的经验的依赖。

7.4 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祖枢老师辛勤的汗水和心血。在此论文完稿之际，首先向李祖枢老师表示崇高的敬意和最诚挚的谢意！在攻读硕士期间，李祖枢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。在李祖枢老师身上，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，同组的李果师兄和董平师弟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！

张 颖

二 OO 八年四月 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[D]. 西南交通大学硕士学位论文. 2006.8.

[2] 谷蓉. 计算机围棋博弈系统的若干问题研究[D]. 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[D]. 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[M]. 上海人民出版社. 1996.

[5] 谢识予. 经济博弈论[M]. 复旦大学出版社. 2002.

[6] 六子棋主页[Z]. http://www.connect6.org/.

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

[8] David N. L. Levy, eds. Computer Games[D]. 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[J]. 电脑学刊 台湾 1990.3.2 :1-8

[10] 张玉志. 计算机围棋博弈系统 [D]. 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][D]. Switzerland: Swiss Federal Institute of Technology (ETH) Z"urich, 1990.

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

[13] 陆汝衿. 人工智能[M], 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[M]. 北京: 机械工业出版社,2000.

[15] Nils J. Nilsson. Artificial Intelligence A New Synthesis [M]. 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[J]. Hsinchu, Taiwan.

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

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

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

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

[22] J. Schaeffer. Distributed Game-tree Searching[J]. 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[J]. Aritificial Intelligence, 1996.

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

Edition[M]. Microsoft Press, 1999.

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

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

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

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

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

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

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

[32] 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 年 06 期.

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

[34] Hall M.R. and Loeb D.E. Thoughts on Programming a Diplomat[J]. Heuristic Programming in Artificial Intelligence 3: the third 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[J].IEEE Transactions on Pattern Analysis and Machine Intelligence, November 1989, 11(11):1203-1212.

[36] MTD 网站. http://theory.lcs.mit.edu/~plaat/mtdf.html[Z].

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

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

[39] 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[J]. 西南师范大学学报（自然科学版）,2007 年 11 月.

[40] 李果．六子棋计算机博弈及其系统的研究与实现[D]. 重庆大学硕士论文. 2007 年 7 月.

[41] 王骄,王涛,罗艳红,徐心和,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[J], 东北大学学报(自然科学版),2005 年第 10 期.

[42] 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.

附 录

A 作者在攻读硕士学位期间发表论文目录

[1] 张颖, 李祖枢. 棋类计算机博弈系统的主要研究方法及其在 6 子棋上的应用.重庆工学院学报（自然科学版）,2008 年 7 月.

[2] 张颖. 6 子棋启发式搜索算法的优化与设计. 西北师范大学学报（自然科学版）, 2008 年 7 月,第 44 卷,4 期.