计算机博弈原理与方法学概述*

徐心和　徐长明
东北大学机器博弈研究室，沈阳，110004
Email: xuxinhe@ise.neu.edu.cn

摘要：计算机博弈伴随着计算机的诞生已经开展了大半个世纪，并以一系列战胜人类天才而享誉世界。计算机博弈在国际上仍在踏踏实实地前进，然而在中国却难有作为。究其原因则是缺少此方面的宣传和基础知识的普及。为此在研究了大量相关资料的基础上，对于计算机博弈原理与方法学进行了概述。从分析棋类游戏属性和博弈过程出发，总结了计算机博弈的关键内容与技术，绘制了博弈软件的基本结构图，介绍了有关数据结构、着法生成、博弈树展开、棋局评估、基本搜索算法、开局库与残局库等相关知识。这是一次探索性的归纳与提升，今后还需要不断地完善和补充。

关键词：计算机博弈 原理与方法学 数据结构 博弈树 搜索算法

Summarization of Fundamental and Methodology of Computer Games

Xinhe Xu　　Changming Xu
Research Group of Computer Games, Northeastern University
Shenyang, 110004, China
E-mail: xuxinhe@ise.neu.edu.cn

Abstract: Computer games have developed for more than a half century since the computer emergence, and made a series of great achievements because the computers beat several human world champions. Computer games are advancing steadfastly in west world, but very hard in China. The reasons are lack of the propagation and popularization of computer games. This paper summarizes the fundamental and methodology of computer games based on studying many interrelated materials and some practice. The work starts from the analysis of game attributes and gaming progress, then sums up the main contents and key technologies, figures a basic structure diagram of computer gaming software, introduces some knowledge on data structure, move generation, expanding game tree, state evaluation, basic search algorithms, opening book, endgame database, and so on. This work is an exploration and upgrade research in the area. So it needs to do more for the consummation and perfect next.

Keywords: compute games; fundamental and methodology; data structure, game tree, search algorithm

1. 引言

计算机博弈（Computer Games），也称之为机器博弈，就是让计算机可以像人脑一样进行思维活动，最终可以下棋，下国际象棋、西洋跳棋、五子棋、中国象棋、围棋等等。

早在计算机诞生的前夜，著名的数学家和计算机学家阿伦·图灵(Alan Turing)便设计了一个能够下国际象棋的纸上程序，并经过一步步的人为推演，实现了第一个国际象棋的程序化博弈[1]。那些世界上最著名的科学家，如计算机创始人冯.诺依曼（John von Neumann），信

*国家自然科学基金资助项目，合同编号（60774097）

息论创始人科劳德.香农(Claude E. Shannon)[2]，人工智能的创始人麦卡锡(John McCarthy)等人都曾涉足计算机博弈领域，并做出过非常重要的贡献。

从上世纪 40 年代计算机诞生，计算机博弈经过一代又一代学者的艰苦奋斗和坎坷历程，终于在上世纪的八、九十年代，以计算机程序战胜棋类领域的天才而享誉世界。其中最为著名的则是 1997 年 5 月 IBM“深蓝”战胜世界棋王卡斯帕罗夫，成为计算机科学史上一个不朽的丰碑。在这之后，计算机博弈一天也没有停息过拼搏。由《Science》杂志评选的 2007 年十大科技突破中，就还包括了加拿大阿尔波特大学的科研成果——解决了西洋跳棋(Checker)博弈问题[3]，也就是说，在西洋跳棋的博弈中计算机将永远“立于不败之地”。

正当世界的目光开始将计算机博弈的方向聚焦在中国象棋、日本将棋和围棋博弈系统的研究与开发的时候，在中国计算机博弈却成了“被爱情遗忘的角落”。寥寥无几的参与者，匮乏的参考文献，沉寂的计算机博弈氛围，使得计算机博弈在中国大陆难有大的作为。其中很重要的原因则是缺少计算机博弈方面得宣传和基础知识的普及，甚至难以找到论述计算机博弈原理与方法学的书籍与资料。

既然提到“原理与方法学”的高度，那就是要研究带有普遍性的、最基本的、可以作为其它规律基础的规律，具有普遍意义的道理，研究在计算机博弈学科上所采用的研究方式、方法的综合。

本文所探讨的计算机博弈还仅仅是局限于完全信息的棋类博弈。由于棋类游戏为广大群众所喜爱，种类与玩法十分丰富。这里在归纳棋类主要属性和类别（第 2 节）的基础上，第 3、4 节分别给出了计算机博弈的基本原理与基本方法，接着（第 5 节）介绍了计算机博弈的重要组成部分——开局库和残局库，在结语一节还论述了计算机博弈的学科性质。应该看到这是一次探索性的归纳与提升，肯定还有不少缺陷与不足，今后还需要不断地完善和补充。

2. 棋类游戏的分类

棋类游戏十分丰富，有些在国际上广为流行，也有些是仅限于地区的民间棋类。而且总会有新的棋类和新的玩法在推出。因此在研究普遍的博弈原理之前有必要对于棋类的属性和分类进行介绍。

2.1 按参与人数分类（Player）：

单人游戏，如华容道等滑块类游戏；

双人游戏，如象棋、围棋、五子棋等。在棋类中双人参与的棋类占绝大多数；

多人游戏，如跳棋。

一般说来，参与人数越多，对手就越多，情况就越发复杂。

2.2 按兵种多少分类（Piece-kind）

单一兵种，如围棋、五子棋、苏拉卡尔塔（Surakarta）、亚马逊（Amazons）等；

多兵种，如国际象棋、中国象棋、日本将棋等。

兵种的增加，意味着着法的复杂程度增加，掌握起来更加困难。

2.3 按着法分类（Move）

走子类：开局前双方摆好，开局后轮流走动棋子。如象棋、国际象棋、跳棋等；

添子类：开局前盘面无子，开局后轮流放入棋子。如围棋、五子棋、六子棋等；

吃子类：对局过程中可以吃掉对方的棋子。如象棋、国际象棋、围棋等；

混合类：在填子的过程中可以吃子（围棋）；在走子过程中可以吃子，还可以填子（日本将棋）。

通常情况弈棋双方轮流施着（招），各走（下）一步。但是有的棋类在一定条件下一方是可以连续施着的，即连续走多步，可成为轮（Turn）。如跳棋、西洋跳棋、黑白棋（Reversi, Othello）、点格棋（Dots and Boxes）等。

2. 4 按判决胜负方式分类（Win-Lose-Draw）

擒获首领：象棋、国际象棋等；

摆成形状：连珠类——井字棋、五子棋、六子棋等；

占领地域：围棋、点格棋等；

剩余子粒：黑白棋、苏拉卡尔塔等；

活动余地：亚马逊等；

到目标地：跳棋、一字棋、牛角棋[4] 等。

3. 计算机博弈的基本原理

3.1 弈棋过程分析

为了深入探讨计算机博弈的原理与方法学问题，有必要分析二人对弈的演化过程，建立相应的数学模型。图 3-1 给出了博弈状态演化过程图[5]。图中表明棋局状态是在着法算子作用下进行演化的，其对应的状态转移方程可以写成

$$S_{n+1} = S_n \cdot q_{n+1}, \quad S_0 = S(0) \tag{3-1}$$

式中 S_0 为棋局的初始局面，q_{n+1} 为第 n+1 步的着法算子，而 S_{n+1} 为下完第 n+1 步后的棋局。于是，不难写出

$$S_F = S_0 \cdot q_1 \cdot q_2 \cdot \ldots \cdot q_F = S_0 \cdot Q \tag{3-2}$$

式中 S_F 为终局，或红胜，或黑胜，或和棋。显然，着法序列 $Q = \{q_1 q_2 q_3 \ldots . q_F\}$ 便是记载博弈过程的棋谱。

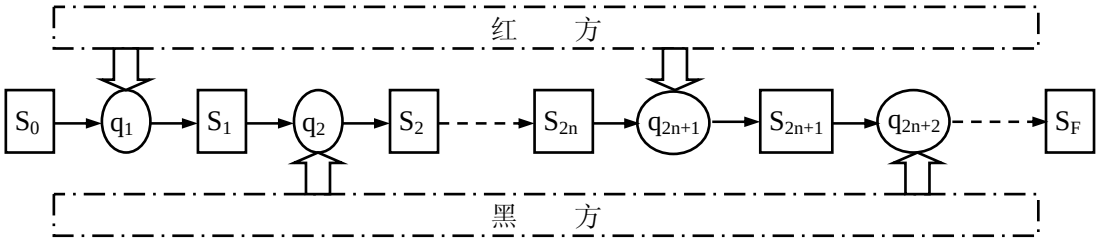

图 3-1　二人博弈状态演化过程图

弈棋的过程是双方轮流给出着法，使棋局向着对本方有利的方向发展，直至最后的胜利。而弈棋的核心是如何给出着法。这是一个复杂的思维过程，简单说来就是：**用着法推演局面，从有利的局面中选择当前的着法。**显然如何正确地评估推演出来的局面则是选手棋力的重要体现。

3.2 如何让计算机下棋？

为了让计算机能够下棋，首要的任务就是通过恰当的数据结构使棋类要素数字化，这里包括：棋盘、棋子、棋规（着法规则，胜负规则）等。

为了用着法推演局面和展开博弈树，就需要具有着法生成器，用以生成该局面

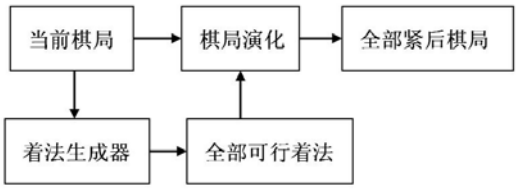

图 3-2 用着法推演局面

下全部（或部分感兴趣）的着法。图 3-2 给出了博弈者思维过程的机器实现过程框图。从而可以产生如图 3-3 所示的博弈树（Game Tree）。节点为局面，树枝为着法，根节点为当前局面，叶节点为展开相应深度的终点局面。双方轮流出手，偶数层节点属于本方（方块表示），奇数层节点属于对方（圆圈表示）。如果叶节点还不是能够给出胜-负-和的最终局面，则要对叶节点进行评估（Evaluation）。以便从有利局面选择当前着法。这便是博弈搜索的职能。

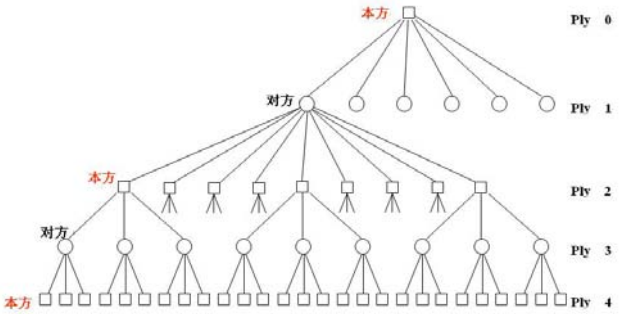

图 3-3　展开深度为 4 的博弈树

搜索引擎根据极大-极小的搜索算法，找到对于本方而言最好的结局和导致最好结局的最佳路径（Principal Variation – 主要变例），从而找到相应的根着法（Root Move）即是本轮搜索所寻求的最佳着法[5]。不难看出，评估和搜索将成为博弈软件的重要部分。

3.3 计算机博弈软件的构成

根据以上的分析，可以给出计算机博弈软件的结构图。如图 3-4 所示。

由于实际的博弈软件经常是采用递归或迭代算法实现博弈树的展开、评估与搜索过程，图 3-4 只是给出了各功能部件之间的关系。

甲方（本方）博弈系统得到当前局面的信息之后，通过复杂的计算过程便可以给出甲方当前的着法。乙方（对方）可以是人，也可以是另外一个计算机博弈系统。双方通过对战平台实现弈棋过程。

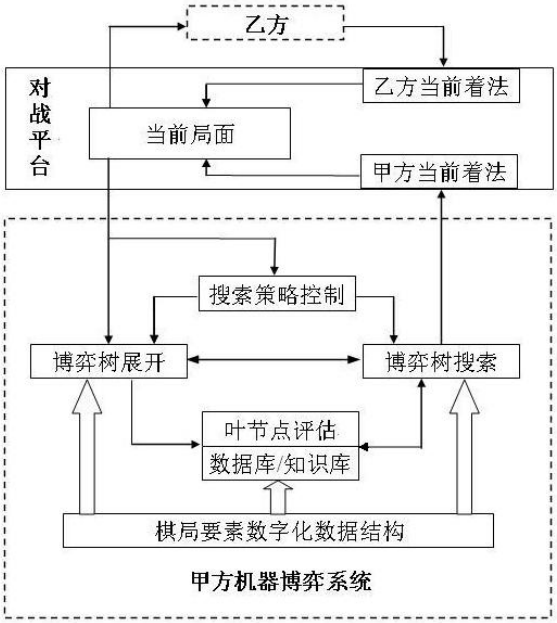

图 3-4　计算机博弈软件结构图

3.4 棋局要素的数据结构

3.4.1 计算机博弈数据结构研究的内容

所有的棋局元素，包括棋盘、棋子、棋局、着法、规则、知识等通过数字化（编码）成为数据元素，而各种数据元素又以特定的关系构成相应的数据结构进行存储和处理。

以最为简单的牛角棋为例，见图 3-5(a)。棋盘似牛角，也像座小山。一枚红子可上可下，力图突破黑方的堵截，走到山下。而两枚黑方棋子，只上不下（可以横走），力图将红子堵回山顶。双方轮流走棋，一方一步。此棋也称之为“娃娃下山”或“娘子下山”，在我国田间地头广为流传。

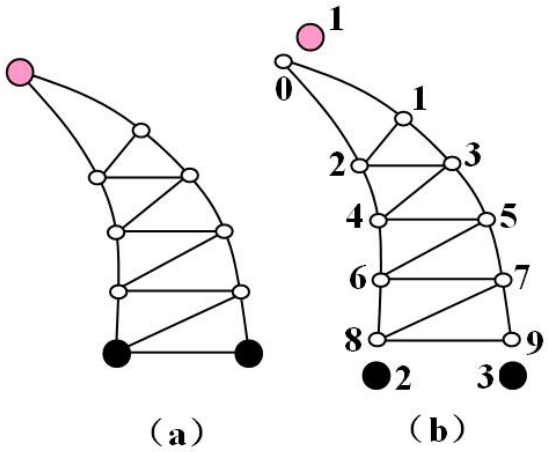

图 3-5 牛角棋编码

设计牛角棋的棋盘和棋子编码如图 3-5(b)所示。10 个棋位编码 0-9，红子 1，黑子 2、3。这样初始棋局便可有两种形式的表示：

（1）棋位向量（1,0,0,0,0,0,0,0,2,3）；

（2）棋子向量（0,8,9）。

如果用 p1,p2,p3 表示红黑 3 枚棋子当前的棋位，则 p1>p2 或者 p1>p3 意味红方获胜；而棋子向量为（0,1,2）或（0,2,1）则意味黑方获胜。[6]

3.4.2 计算机博弈数字化原则与意义

数据结构问题在计算机博弈中的重要性是不言而喻的。在展开和搜索博弈树期间，数以千百万计的着法和棋局被生成、存储、撤销，合理的数据结构可以显著地提高搜索速度和深度，节省内存空间，改进博弈效果。数据结构还对编程有着直接的影响。以图 3-5(b)为例，如果将左右两侧节点编码互换，则要给编程带来很大的麻烦。

在采用宽度优先的搜索算法中，新被扩展的节点应该采用队列结构；而在深度优先的搜索算法中，则应该采用栈结构。

显然编码问题要面向棋种、面向算法、面向编程。许多情况下适当的冗余是必要的。比如牛角棋棋局的棋位向量和棋子向量同时存在，避免频繁的相互的转换，会对棋局评估和着法生成带来许多的便利。

3.4.3 哈希技术与哈希表

在博弈过程中是经常需要比较两个局面是否相同。如果是比较每个棋子的位置，或许不需要花很多时间，但是实战中常常需要比较的局面多得惊人（数以十万百万计），如在开局库中寻找对应的局面。于是这一操作的时间和空间开销便成为博弈搜索的瓶颈。

众多棋类的成功经验表明，棋局的存储最好是采用 Zobrist 哈希技术加以实现[7]，该技术能很容易地将棋局映射为哈希数（Hash Number）。它的基本原理是先将各个棋子的代码及其坐标位置分别映射为一个大的（如 64 位或 32 位等）伪随机数，再将棋盘上全部棋子所对应的随机数进行异或求和运算，最终得到的哈希数便作为该棋局的索引值（Zobrist 键值），用作棋局的存储与查询。

Zobrist 哈希数的最大优点就在于它可以增量计算。在着法的作用下，棋局中的增子、减子或挪子操作，都能统一地用棋子变化所对应的哈希数和表示局面的哈希数做简单的异或运算来完成。

博弈树展开过程中，会产生大量的因不同着法顺序而导致的相同局面（重复节点）。要避免不必要的重复计算，就要把搜索过的节点随时存储起来，这就需要一种称为置换表（Transposition Table）的数据结构，一般实现为一个哈希表(Hash Tables)。通常哈希表可以避免重复节点的重复展开。此外，哈希表还常保存一些和局面密切相关的重要信息，例如，最佳着法、是否受到严重的威胁等。即或需要对该节点重复搜索时，存储在哈希表中的相关信息还会对展开的子树有良好的启发功能。[8]

3.4.5 比特棋盘（Bit Board）

在着法生成和棋局评估的过程中，时常仅仅关心一些棋子的分布，这时可以用比特棋盘（亦称位棋盘）表示棋子的某种状态，它其实是棋子状态条件的布尔表示。如棋盘中哪些棋位上有红子？哪些棋位上有黑车等等。比特棋盘的定义为

$$B = [b_{i,j}]_{m\times n}, \qquad b_{i,j} = \begin{cases} 1 & s_{i,j} = true \\ 0 & s_{i,j} = false \end{cases} \tag{3-3}$$

式中 $s_{i,j}$ 为棋位（i,j）的布尔条件。

3.5 棋局评估[9]

如果在叶子节点不能给出胜-负-和的结果，那么评价这样的局面对我“有利”还是“不利”，以及 “有利”或“不利”的程度，则只能依靠评估函数了。通常设计局面的评估函数需要考虑如下不同类型的知识，并通过量化后加权组合而成。

3.5.1 子力（Material）

在象棋和国际象棋中，它是所有子力价值的和。在围棋或黑白棋中，通常计算双方棋盘上棋子的数量。但是黑白棋有个有趣的反例：棋局只由最后的子数决定，而在中局根据子力来评价却是很差的思路，因为好的局势下子数通常很少。其他像五子棋一样的游戏，子力是没有作用的，因为局面好坏仅仅取决于棋子在棋盘上的相互位置，看它是否能够发挥作用。

3.5.2 位置（Position）

棋子落于不同的棋位其作用可能差别很大。象棋中的车占中路，兵过河，马卧槽都是具有威胁性的位置。相反如果马窝心，兵下底又都不甚理想。围棋中的星位也是兵家必争之地。于是不同的位置给予不同的分值，以表示不同的价值。

3.5.3 空间（Space）

在某些棋类中，棋盘可以分为本方控制的区域和对方控制的区域，以及有争议的区域。在围棋中，这个思想被充分体现。而包括象棋在内的一些棋类也具有这种概念，本方的区域包括一些棋位，它被本方的棋子攻击或保护，而不被对方棋子攻击或保护。在黑白棋中，如果一块相连的棋子占据一个角，那么这些棋子就吃不掉了，成为该方的领地。空间的评价就是简单地把这些区域加起来。如果所含棋位的重要程度存在差别，那就在区域的计算上增加棋位重要性的因素。

3.5.4 机动性（Mobility）

各个棋子的机动性如何，关系到棋子可行着法的多少。如象棋中的马是可以“马踏八方”的，但是贴边或被憋腿，其活动余地大减。显然机动性越好，可行着法越多，选择有利局势的机会也越多。

3.5.5 拍节（Tempo）

某些情况下起决定作用的不是着法的多少，而是出招的拍节和数量的奇偶性等。以一数学游戏为例：有两堆石子，双方轮流从石堆中拿去几颗，每次只能从一堆石子中拿走至少一颗石子，拿完最后一堆者获胜。这个游戏的诀窍是：始终让对方面临两堆石子一样多的窘境。面向这样的问题时，先手方只要头一步让两堆石子数目一样多就可以了。然后对手从某一堆取走几颗，先手方便在另一堆取走同样数量的石子。在黑白棋和象棋中都会遇到这样的问题。而对于一字棋和点格棋这着则是制胜的法宝。

3.5.6 威胁（Threat）

威胁的思想是“步步紧逼”，或要取胜，或要吃子，使对方仅有招架之功，而无还手之力。在五子棋、六子棋中常常采用基于威胁的搜索，即不断地摆出强迫对方防守的棋型，成为取胜的必由之路。

3.5.7 形状（Shape）

形状的好坏对于局面的影响一般是长远的，在浅层的搜索中不易发现。在象棋中，对手的空头跑，单车栓车马，就都是不好的形状，最终使本方处于被动的局面。对于连珠棋一类通过“摆成形状”而决定胜负的棋类则要计算各种棋型的数量，于是形状就成为评估的焦点。

3.5.8 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。许多图案在围棋中称之为模式，如 3×3 上下文模式，通过对大量高手对局中的模式提取和出现频率的统计，构造模式库，便可以由此提供下一手落子的最佳位置。

3.5.9 棋局性能评估

从表面上看，棋局评估的越全面、越准确，棋力性能就会越高。其实不然。一般说来，

棋力性能 = 知识×速度，时间作为约束条件。评估中考虑的问题越多、越细致，耗费的时间则越多，必然影响到单位时间内搜索节点的数目，影响到搜索的速度和深度。

在博弈软件的设计中通常存在“重知识”（评估）和“重速度”（搜索深度）两种倾向，并且都有成功的范例。当然能够权衡利弊，将两者有机结合则是最为理想的方案。这也是目前共同努力的方向。

3.6 博弈树展开与分析

博弈树是由树枝和节点构成单向无环图。树枝是着法，节点是由该着法生成的局面。因此博弈树展开的过程就是着法生成的过程。

3.6.1 着法生成的不同策略[8]

（1） 选择生成。根据当前的棋局，只生成部分可行的着法，而不去考虑其它可行着法。比如象棋在被将军的局面下，仅需要考虑“避将着法”，即摆脱被将军状态的着法；

（2） 渐进生成。先产生一些着法，并沿着某个着法延伸下去，直到证明这条路线坏到足以中止搜索的程度，再去生成和搜索其它的着法。比如象棋中先生成“吃子着法”，然后再考虑“非吃子着法”；

（3）完全生成。一次产生所有的着法。显然这是一种稳妥而保守的做法。

3.6.2 走子类着法生成方法[5]

（1）棋盘扫描法（含射线法）。根据棋规，在当前棋局中逐一找到棋子可行的落址。该方法最为直观，编程也很简单。但是由于需要反复在棋盘上扫瞄，时间开销巨大，一般缺少实战意义。

（2）模板匹配法。对于有特殊要求的走子类棋子，如象棋中的马和象，将根据走子规则“绘制”的模板套在棋子当前的位置上，通过模板找到可行的落址，形成可行着法。

（3）预置表法。这是一种用空间换时间的生成策略。将全部棋子在所有棋位上的着法都预先放在表中。开局时自动生成，放入内存。弈棋过程中直接查找。

以牛角棋为例，红黑棋子的全部着法存放在 3 维的预置表 int preTable 中，见图 3-6。第 1 维区分红黑棋子，第 2 维对应 10 个棋位，第 3 维给出在该棋位上的可能着法落址，INV 为着法队列结束符。这样就可以根据棋子当前所在的位置直接查出全部可能的着法。当然还要检查该落址是否是空位。

```
int preTable[2][10][5] = { {    /*red stone moves table*/
      {2, 1, INV},{2, 3, 0, INV}, {4, 3, 1, 0, INV},{5, 4, 2, 1, INV}, {6, 5, 3, 2, INV},
      {7, 6, 4, 3, INV},{8, 7, 5, 4, INV},{9, 8, 6, 5, INV}, {9, 7, 6, INV},{8, 7, INV}, },
                              {  /*black stone moves table*/
      {INV},{0, INV},{3, 1, 0, INV},{2, 1, INV},{2, 3, 5, INV},{3, 4, INV},
      {4, 5, 7, INV},{5, 6, INV},{6, 7, 9, INV},{7, 8, INV}, },    };
```

图 3-6 牛角棋着法预置表

3.6.3 添子类着法生成方法[10]

一般说来，添子类的着法生成比较直观，即盘面上的合法空位便可以落子。其中五子棋和六子棋最为简单，下完的棋子不再会改变。黑白棋稍复杂些，下完的棋子可能会被后续着法所变换黑白，但每下一子棋盘上就多一子。围棋是最复杂的，由于存在提子的着法，所以局势是可逆的，对于打劫这样的着法还需要更为复杂的处理过程。

3.6.4 博弈树展开与分析

一颗完整博弈树的规模是相当可观的天文数字。中国象棋完整博弈树的节点数高达 10^{150}，据说地球上全部原子的数目也才有 10^{132}。显然是无法全部展开和进行遍历搜索。即

使选用搜索速率为 1M 节点/s 的计算机系统，日夜不停地搜索 100 年，也才只能搜索 9 层。还达不到一般象棋大师的水平。即或是最简单的牛角棋，如果将博弈树展开 15 层，还不将无意义的循环走棋的节点计算在内，那有效博弈树的总节点数还接近 750 万个。

需要特别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。在图 3-3 中，本方走棋时，它在偶数层的着法选择是要在其全部子节点中找到评估值最大的一个，即实行“Max 搜索”。而其应对方在奇数层的着法选择则是在其全部子节点中要找到评估值最小的一个，即实行“Min 搜索”。如果没有特殊的搜索方法那将是非常棘手的问题。幸好香农(Claude Shannon)教授早在 1950 年就提出了“极大-极小算法”（Minimax Algorithm）[2]，从而奠定了计算机博弈的理论基础。

4. 计算机博弈求解的基本搜索方法

4.1 优化搜索与博弈搜索

搜索是一个非常宽泛的概念，也是有着广泛应用领域的技术。

在各种规划问题中，如线性规划、非线性规划、整数规划等；在许多优化问题中，如旅行商问题、计划调度问题等，在无法得到解析解（含数值方法求解）的情况下，便会采取各种启发式搜索算法。如爬山法、分支定界法、禁忌搜索、模拟退火、遗传算法、蚁群算法等等。虽然目前求解此类优化问题的搜索算法很多，但是他们有着如下共同的特点：

（1）单一决策主体，即统一的性能指标函数；

（2）明确的目标函数和约束条件，通常可以用数学模型来描述；

（3）基本为静态规划和优化问题，亦即单步决策问题。

计算机博弈属于博弈和对策的范畴，是由两个非合作主体构成的、多步的、动态博弈问题，对弈双方有着对立的、二人零和的决策目标，其目标函数难以用数学模型来描述。目前能够用以描述的主要是博弈树。而普遍采用的求解方法就是在博弈树中搜索。博弈搜索的目标就是搜索最佳路径，搜索当前的最佳着法（根着法），并且亦步亦趋地进行下去。

博弈搜索从搜索方向上可以分为宽度优先搜索(Breadth-first search)和深度优先搜索(Depth-first search)。前者能够保证在搜索树中找到一条通向目标节点的最短途径。但是巨大的时间和空间开销使得它在计算机博弈中很少采用。深度优先搜索的最大特点就是可以节省大量的节点存储空间。因为它通常是采用递归过程来遍历搜索树，即后序遍历，得到一个节点值，就对其子节点做递归，然后根据他们的返回值来决定自身的返回值，搜索过的底层节点也随即撤销，因此它所占用的动态空间十分有限。Alpha-Beta 剪枝和置换表相结合的算法更使得博弈树的规模被数量级地减少。在最好的情况下，它处理的节点数是纯粹的最小-最大搜索（Min-Max Search，亦称极大-极小搜索）的平方根的两倍，可以从 15,000,000 个减少到 7,800 个。[10]

4.2 基于 Alpha-Beta 剪枝的搜索[11]

目前以国际象棋为代表的绝大多数博弈程序的搜索算法都是在“带置换表的启发式 Alpha-Beta 搜索”基础上发展起来的，这就涵盖了极大-极小搜索、Alpha-Beta 搜索、迭代加深和置换表四方面内容。有关置换表（TT）的概念已经在哈希技术与哈希表（3.4.3 节）做了简单的介绍。这里仅对剪枝算法和基本搜索的概念进行必要的阐述。

4.2.1 极大-极小搜索（Min-Max Search）

由于对弈双方都很理智，都想赢棋，在考虑着法的时候都尽量想让棋局朝着自己的方面转化，所以在同一颗博弈树上（图 3-3），在不同的层面（Ply）上就要有不同的选择标准。这也就是称之为“变性”搜索树原由。在偶数层节点的着法选择是要在其全部子节点中找到评估值最大的一个，即实行“Max 搜索”。而在奇数层节点的着法选择则是在其全部子节点中要找到评估值最小的一个，即实行“Min 搜索”。

在进行极大-极小搜索的时候，首先要在有限深度内展开全部叶子节点，并进行评估。然后自下而上地进行搜索计算，奇数层节点取其子节点估值的极小值，偶数层节点取其子节点估值的极大值，一直反推算到根节点。在反推的过程中始终要记住算出该值的子节点是谁，这样就可以得到一个从根节点到叶子节点的一条路径，这就是“最佳路径”，它是双方表现最佳的对弈着法序列。文献中称其为主要变例（PV）。而主要变例的根着法便是本方当前应该选择的着法。

4.2.2 Alpha-Beta 搜索（Alpha-Beta Search）

Alpha-Beta 搜索是基于深度优先的搜索。考虑到极大-极小搜索的特殊性，引进了 Alpha-Beta 剪枝技术，使得搜索的效率显著提高。

如果博弈树的局部为偶层-奇层-偶层的关系，如图 4-1 例题所示，在搜完左边一枝之后，父节点得到一个值 4，可以称之为 Alpha 值；如果在接续支路的叶子节点上发现了一个小于 Alpha 的节点，则整个支路便可以剪掉，即没有必要再搜下去。因为根据极小-极大的运算关系，这一枝不可能对局面有更好的贡献。不难看出，Alpha 值始终是本方最佳着法的下界。

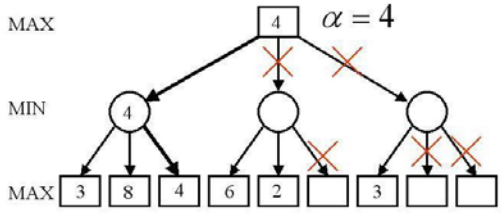

图 4-1 Alpha 剪枝示意图

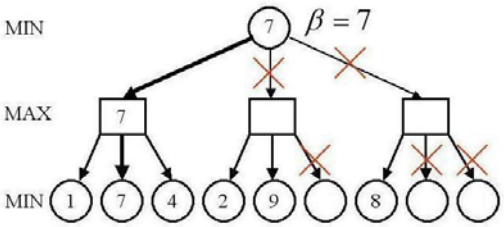

图 4-2 Beta 剪枝示意图

Beta 剪枝出现在奇层-偶层-奇层的子树部分。如图 4-2 例题所示，在搜完左边一枝之后，父节点得到一个值 7，可以称之为 Beta 值；如果在接续支路的叶子节点上发现了一个大于 Beta 的节点，则整个支路便可以剪掉，即没有必要再搜下去。因为根据极大-极小的运算关系，这一枝不可能对局面有更好的贡献。不难看出，Beta 值始终是本方最佳着法的上界。同时也是对手所能承受的最坏的结果，因为在对手看来，他总是会找到一个对策不比 Beta 更坏的。

于是由 Alpha-Beta 值构成的窗口便成为最佳着法的取值范围，称其为 Alpha-Beta 窗口。

4.2.3 负极大值算法（NegaMax Algorithm）

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[12]。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。

$$F(v) = -\max\{-F(v_1), -F(v_2), \cdots, -F(v_n)\} \qquad (4\text{-}1)$$

其中 $F(\cdot)$ 为节点估值，$v_1, v_2, \cdots, v_n$ 为节点 v 的子节点。

此时需要特别注意的则是，如果叶节点是红方（本方）走棋，评估函数返回 *RedValue-BlackValue*，如果是黑方（对方）走棋，则返回 *BlackValue -RedValue*。另外，由于负极大值计算等价于“Min 搜索”，所以这里仅进行 Beta 剪枝，不再有 Alpha 剪枝。

4.2.4 迭代加深搜索（Iterative deepening search）[13]

在对 alpha-beta 剪枝算法进行深入研究之后，就会发现剪枝效果是和着法排序密切相关。如果一开始就能从最佳路径（PV）展开搜索，一定会得到一颗最小的博弈树。显然这是不可能的，因为搜索的目标才是最佳路径。但是不难想像，深度为 D－1 层的 PV_{D-1}，最有可

能成为深度为 D 层博弈树的 PV。于是人们便提出了迭代加深的搜索算法。先搜索 1 层，找到最佳着法，再搜 2 层的博弈树，找到 2 层的 PV_2；进而搜索 3 层博弈树，得到 PV_3；逐渐加深，直到 PV_{D-1}；最后实现对于 D 层博弈树的搜索。

迭代加深搜索的优点在于：

（1）在时间受限的情况下，不知道可能搜索多深，此策略可在时限到达时即刻结束搜索，并能给出较好的结果；

（2）显著提高剪枝效果。表面上看，迭代增加了搜索的遍数，“浪费”了一些时间。但是有效的剪枝可以显著地减少分支因子，从而提高了搜索效率。

图 4-3 给出了迭代加深搜索程序流程图。其中历史表记载着相应局面下的最佳走法。[8]

```
for (i = 1; i < MAX_DEPTH; i ++) {
  AlphaBeta(-INFINITY, INFINITY, i);
  if (超过最短搜索时间) {
    break;
  }
}

int AlphaBeta(int vlAlpha, int vlBeta, int nDepth) {
    if (nDepth == 0) {
      return 局面评价函数;
    }
    生成全部走法;
    按历史表排序全部走法;
    for (每个生成的走法) {
      走这个走法，生成子节点局面;
      int vl = -AlphaBeta(-vlBeta, -vlAlpha, nDepth - 1);
      撤消局面;
      if (vl >= vlBeta) {      / Beta 剪枝
          将这个走法记录到历史表中;
          return vlBeta;
      }
      if (vl > vlAlpha) {
          设置最佳走法;
          vlAlpha = vl;
      }
    }
    将最佳走法记录到历史表中;
    if (根节点) {
      最佳走法就是电脑要走的棋;
    }
    return vlAlpha;
}
```

图 4-3 迭代加深程序流程图

4.3 基于蒙特卡洛模拟的博弈树搜索

蒙特卡洛树搜索（Monte-Carlo Tree Search, MCTS）是一种最佳优先搜索（Best-first search）算法，更适合于分支因子很大的博弈树搜索。比如围棋分支因子常常大于100，而亚马逊棋的分子因子可以大于1000，采用前述的Alpha-Beta搜索算法只能搜索很浅的几层，导致棋力水平难以提高。自从Bernd Brugmann等首先将蒙特卡洛模拟技术用于围棋的动态评估[14]，进而出现把UCB1（Upper Confidence Bound）算法[15]扩展到树搜索的UCT（UCB applied to Tree）算法[16]，使得在短短的几年中，九路围棋的计算机程序已经可以和围棋高手对弈了。

蒙特卡洛模拟对局就是从某一棋局出发，随机走棋。有人形象地比喻，让两个傻子下棋，他们只懂得棋规，不懂得策略，最终总是可以决出胜负。这个胜负是有偶然性的。但是如果让成千上万对傻子下这盘棋，那么结果的统计还是可以给出该棋局的固有胜率和胜率最高的着法。

蒙特卡洛树搜索通过迭代来一步步地扩展博弈树的规模，UCT 树是不对称生长的，其生长顺序也是不能预知的。它是根据子节点的性能指标导引扩展的方向，这一性能指标便是 UCB 值（公式 4-2）。它表示在搜索过程中既要充分利用已有的知识，给胜率高的节点更多的机会，又要考虑探索那些暂时胜率不高的兄弟节点，这种对于“利用”（Exploitation）和“探索”（Exploration）进行权衡的关系便体现在 UCT 着法选择函数的定义上，即子节点 N_i 的 UCB 值按如下公式计算。

$$\overline{X_i} + C\sqrt{\frac{\ln t(N)}{t(N_i)}} \qquad (4\text{-}2)$$

式中 N 为给定节点，N_i 为其子节点，$t(N)$为对 N 节点的模拟次数，$t(N_i)$为 N_i 节点被选中的模拟次数，C 为加权系数。$\overline{X_i}$ 为子节点 N_i 的收益平均值。可见 UCB 公式由两部分组成，其

中前一部分就是对已有知识的利用，而后一部分则是对未充分模拟节点的探索。*C* 小偏重利用；而 *C* 大则重视探索。需要通过实验设定参数来控制访问节点的次数和扩展节点的阈值。

蒙特卡洛树搜索（MCTS）仅展开根据 UCB 公式所计算过的节点，并且会采用一种自动的方式对性能指标好的节点进行更多的搜索。具体步骤概括如下：

1. 由当前局面建立根节点，生成根节点的全部子节点，分别进行模拟对局；
2. 从根节点开始，进行最佳优先搜索；
3. 利用 UCB 公式计算每个子节点的 UCB 值，选择最大值的子节点；
4. 若此节点不是叶节点，则以此节点作为根节点，重复 2
5. 直到遇到叶节点，如果叶节点未曾经被模拟对局过，对这个叶节点模拟对局；否则为这个叶节点随机生成子节点，并进行模拟对局；
6. 将模拟对局的收益（一般胜为 1 负为 0）按对应颜色更新该节点及各级祖先节点，同时增加该节点以上所有节点的访问次数；
7. 回到 2，除非此轮搜索时间结束或者达到预设循环次数；
8. 从当前局面的子节点中挑选平均收益最高的给出最佳着法。

由此可见 UCT 算法就是在设定的时间内不断完成从根节点按照 UCB 的指引最终走到某一个叶节点的过程。而算法的基本流程包括了选择好的分支（Selection）、在叶子节点上扩展一层（Expansion）、模拟对局（Simulation）和结果回馈（Backpropagation）这样四个部分。[17]

UCT 树搜索还有一个显著优点就是可以随时结束搜索并返回结果，在每一时刻，对 UCT 树来说都有一个相对最优的结果。

对于围棋一类十分复杂的搜索对象，即或进行数百万次的模拟对局也只能覆盖整个博弈树的很小一部分。因此完全随机地选择子节点将会延缓最优解的收敛速度，需要在子节点的随机选择的过程中加入基于围棋知识的各种策略，也就是根据这些策略不同的重要程度赋予不同的随机性，从而可以得到更高的搜索效率和和更好的结果。这里同样也存在如何权衡知识与速度之间的矛盾问题。

5. 开局库与残局库

5.1 开局库

开局库几乎是每个棋类博弈程序必备的部件，它的好处在于：即使是再笨的程序，开局库能使得它在开局阶段看上去不那么业余。既能防止因为复杂的搜索而浪费大量时间，也能防止在开局阶段犯下战略性的错误。[8]

开局库是将高手开局时的棋谱记载下来，仿照执行。由于开局形式多种多样，局面常常数以千万计。此时必须应用哈希技术才能方便存储和查找。

对应于同一局面常常可以有多种着法，此时应该统计各种着法的胜率，作为随机选择时的依据。开局库还应该具有自学习的功能，可以随着程序见识的增长，补充新鲜的定式，或调整已有着法的胜率参数。

5.2 残局库

残局库却不一定是博弈程序必备的部件。只有对于吃子类棋类，当盘面剩余子力不多进入残局时，仍然按照中局的搜索算法难以取得良好的效果。原因在于：

（1）评价指标发生变化。多数中局的目标是“谋取优势”，那在残局阶段则应该是：优势谋胜，劣势谋和，均势谋机。因此如何“谋胜”、“谋和”的策略显得十分重要。于是评价体系需要调整；

（2）许多局面致胜的途径十分特殊和单一，没有足够深度的搜索是难以发现的。

因此如何通过“学习算法”将博弈大师丰富的残局知识存放到博弈系统中去，还是很有

挑战性的研究课题。目前国际象棋根据回溯算法推演出子数有限的残局库，但对于中国象棋而言还有许多工作要做。[18, 19]

6. 结语

计算机博弈原理与方法学既涉及到博弈论（对策论）、搜索原理等理论内容，又更多地涉及到数据结构、软件工程、程序设计方法学等方面的知识。计算机博弈属于计算机科学与应用学科的研究方向之一，又是人工智能领域的重要研究方向，应该属于智能科学与技术学科的一个分支。虽然这一方向的研究工作取得了举世瞩目的成果，但是相关的理论与应用技术的归纳与提升还有很多工作要做。这也是把计算机博弈技术应用到其它领域所不可或缺的基础性工作。本文尚属一种尝试性的阶段性成果，今后还需要不断充实与完善。

致谢：象棋百科全书网站站长黄晨先生提供了丰富的资料，不仅给本文的研究以很大的帮助，在中国计算机博弈领域的影响也是有目共睹的，在此一并表示感谢。

参考文献

[1] Frederic Friedel, michael 译注. 电脑国际象棋简史. http://www.chessit.net/file_topic/computerchess/c_briefhistory.htm

[2] Shannon, Claude E., Programming a computer for playing chess[J], Philosophical Magazine, Vol. 41:256-275, 1950.

[3] Science-10-Breakthrough of the Year2007, http://www.sciencemag.org.cn, 22. Dec. 2007

[4] 罗鉴江，民间棋类游戏[M]. 北京：农村读物出版社. 2003.9

[5] 徐心和，王骄. 中国象棋计算机博弈关键技术分析[J]. 小型微型计算机系统，2006 ,27(6)：961-969

[6] 徐心和，王浩，孔凡禹. 事件对策理论及在棋类游戏中的应用[J]. 2007 年中国智能自动化会议论文集，2007.8.，兰州，中南大学学报（增刊），24-27

[7] Zobrist A. A new Hashing Method with application for game playing[R]. Technical Report88, Computer Science Department, University of Wisconsin, Madison. 1970.

[8] 黄晨. 电脑象棋循序渐进. http://www.elephantbase.net/computer/stepbystep3.htm

[9] David Eppsten，黄晨译注. 对弈程序基本技术—评价函数. http://www.elephantbase.net/computer/evalue_intro1.htm

[10] François Dominic Laramée，黄晨译注. 国际象棋程序设计. http://www.elephantbase.net/computer/basic_started.htm

[11] 王小春，PC 游戏编程[M]. 重庆：重庆大学出版社, 2002

[12] D.E Knuth and R.N Moore，An analyze of Alpha-Beta pruning[J], Artificial Intelligence, Vol,6,1975, pp.293-326

[13] R E Korf, Depth-first iterative-deepening: An optimal admissible tree search [J]. Artificial Intelligence, 1985, 27(1): 97-109

[14] Bernd Brugmann, Fohringer Ring 6. Monte Carlo Go. http://www.ideanest.com/vegos/MonteCarloGo.pdf

[15] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem [J]. Machine Learning, 47(2/3):235–256, 2002.

[16] L. Kocsis and C. Szepesvari. Bandit based monte-carlo planning[C]. In 15th European Conference on Machine Learning (ECML), 282–293, 2006.

[17] Guillaume Chaslot, Sander Bakkes, Istvan Szita and Pieter Spronck. Monte-Carlo tree search: A new framework for game AI[C]. Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference

[18] Wu R, Beal D F. A memory efficient retrograde algorithm and its application to Chinese chess endgames [J]. More Games of No Chance MSRI Publications Volume 42,2002, 207-228

[19] H.-r. Fang, T.-s. Hsu, S.-c. Hsu, Construction of Chinese Chess endgame databases by retrograde analysis[C], in: T.A. Marsland, I. Frank (Eds.), Computers and Games 2000, Lecture Notes in Computer Science, Vol. 2063, Springer, New York, 2001, 99–118.