人机对弈的五子棋 AI 程序

吴佳龙
软件 83
2018013418
wu-jl18@mails.tsinghua.edu.cn

赵伊书杰
软件 83
2018013394
zysj18@mails.tsinghua.edu.cn

ABSTRACT
本次实验使用 Minimax 搜索算法及 alpha-beta 优化剪枝，并结合了多种优化技术（柱搜索、散列表、迭代加深搜索等），实现了可进行人机对弈的五子棋 AI 程序。通过比较搜索的平均等效分支因子，我们验证了我们的搜索优化技术能有效提高搜索的效率。在 Botzone 在线平台上与其他五子棋 AI 进行博弈，说明了我们实现的 AI 具有较强的五子棋棋力。

Keywords
五子棋, 人工智能, Minimax 搜索, alpha-beta 剪枝.

1. 简介
无禁手五子棋在大小为 15 × 15 的棋盘上，黑子先行，轮流于棋盘空点处，先把五枚或以上已棋相连成任何横纵斜方向为胜。

本次实验首先使用 Minimax 搜索算法及 alpha-beta 剪枝，结合柱搜索和局面评估函数技术，实现了最基本的五子棋落子位置的搜索（算法 1）；并在算法 1 的基础上加入了散列表（在博弈搜索中称置换表或换位表）优化方法，避免重复搜索相同局面（算法 2）；最后，加入了迭代加深方法，通过深度更浅的搜索结果引导当前行棋排序，进一步增加 alpha-beta 剪枝的效率（算法 3）。

通过记录搜索的深度和结点数，我们能够计算一次搜索的等效分支因子。比较平均等效分支因子，能够验证我们在 alpha-beta 搜索基础上的优化能够提高搜索的效率。

我们将我们的 AI 托管在 Botzone 平台[1]的无禁子五子棋游戏上，用于测试 AI 的棋力强弱。我们与平台上其他五子棋 AI 进行对战，并获得天梯积分。较高的稳定上涨的积分与排行榜排名，能够证明，我们的搜索和优化方法能够有效地实现一个较强的五子棋 AI。

此外，我们还在已有代码框架的基础上完善了悔棋、复盘等功能。

2. 数据结构与变量设计说明
2.1 棋盘状态
在头文件 define.h 中，定义了表示当前棋局状态的全局变量：二维数组 chessBoard[GRID_NUM][GRID_NUM]存储当前棋盘每个位置的着棋状态（黑或白或空）；winner 记录赢方；agent 和 user 分别记录 AI 和用户执黑或者执白；moveTrace 记录了每一方每一步的落子位置；timeStamp 记录当前棋局步数（时间）；remainBlank 记录场上剩余可用的落子点，其初始值为 225。

结构体 point 是一个简单的 int 对，用于表示棋盘上的一个位置。

2.2 散列表
在头文件 hash.h 中，定义了对局面进行 hash 所需的 Zobrist 键值以及 hash 表结点的结构体。

zobristValue[GRID_NUM][GRID_NUM][3]存储了每个位置的着棋状态的 Zobrist 键值，whiteFirstValue 和 MinFirstValue 分别存储了白棋先手和用户先手的键值。它们用于生成整个局面的 Zobrist 键值用于 hash。

结构体 hashNode 是散列表的结点，它包含有成员 zobrist, depth, flag, value, move, time，分别表示局面 Zobrist 值，搜索深度，估值标签，估值，已知最优移动，最近一次访问或更改该结点的时间。

Hash 算法描述详见 3.3 节。

2.3 局面估值
在头文件 evaluate.h 中，定义了用于局面估值的权重，它们是 cheng, chong, xianShouCheng, xianShouChong, PosValue 等。

局面估值算法以及对这些权重的介绍详见 3.3 节。

3. 算法描述
3.1 Gameover 模块
Gameover 模块中的两个重载函数 gameover 分别用于判断某一方在某点落子是否会使得棋局结束，以及当前整个局面是否结束并返回获胜方。

一场对局的状态可能为：AI 胜利、玩家胜利、平局。其检测方法如下：

gameover()是程序终止前调用的函数，它会返回准确的对局状态，但是会带来大量时间开销。其工作原理为，首先查看 remainBlank 是否为 0，若是，直接返回平局。否则，算法会扫描棋盘中的每一行、列、斜行，并查看是否有五个连在一起的相同颜色的棋子，如果有，则该颜色对应的玩家胜利。

[1] https://www.botzone.org.cn/game/Gomoku；由北京大学信息科学技术学院人工智能实验室开发

如果既不是平局，也没有获胜方，则对局仍在进行，没有中止。

gameover(pos, player)是为了搜索函数设计的快速判断函数。在 alpha-beta 搜索中，搜索函数只关心 player 是胜利还是平局就可以做出决策，而不关心其它状态。因此，该函数只会先通过 remainBlank 判断在 pos 处下棋是否会导致平局，而后再通过计算 pos 是否会和附近的棋子连成五个相同的棋子，判断 player 是否会获胜。这个函数能以较低开销满足搜索函数的需求。

3.2 Createmoves 模块

3.2.1 柱搜索

柱搜索是一种**前向搜索**技术。前向搜索指在某一个结点上无需进一步考虑而直接地剪枝掉一些子节点；柱搜索在每个结点指考虑最好的 k 步行棋可能。（**本次实验中我们取** k=8；行棋的估值见 3.3.2 节）

在五子棋中，一般局面可行的落子点较多，而真正有效的落子实际不多，因此我们需要柱搜索技术。

3.2.2 柱搜索的优化实现

柱搜索主要使用单步估值算法来评价每个可用落子点的效果。根据测试，还加入了下面两个优化来提高剪枝的效果：

1. 使用 Zobrist Hash 加速生成扩展集

   算法会保留一部分局面已经生成的扩展集。下次遇到这些局面时，会直接返回以前生成的扩展集，加速搜索。

2. 在扩展集中加入防守落子点

   在前期版本中，单步估值算法（详见 3.3.2 节）给出的评估常常会使得 AI 只能进攻，不能防守。因此，我们令算法从对手的角度再进行一次评估，选出对手的若干个最佳进攻点。而后将其作为 AI 的防守落子点，加入到扩展集中去。

3.3 Evaluate 模块

Evaluate 模块中包含两个全局估值函数 Evaluate、evaluate，分别用于完整评估某一方的局面估值；还包括单步估值函数 evaluateStep，用于快速为柱搜索生成最佳扩展集（见 3.3.1 节）；而 patternAnalysis 是上述三个函数均会调用的函数，它被用来计算组合棋型的估值（详见 3.2.1.2 节）。

3.3.1 局面估值

算法通过搜索局面中的单线棋型、组合棋型来计算估值。随后，估值函数返回权值和，并以此来指导搜索算法下一步的行动。下面具体介绍这两种棋型。

3.3.1.1 单线棋型的权值

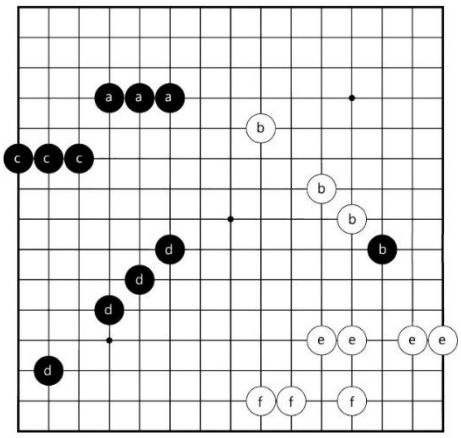

图 3-1 单线棋型举例

五子棋的基本棋型分为**成**和**冲**两种。成指的是出现了若干个子相连的情况。比如，图 3-1 中的 a 棋型显示了三个子相连的情况，称为“成三”。冲指的是两个同行（列、斜行）的成之间，恰好隔了一个空位的棋型。比如，图 3-1 中的 b 棋型显示了成一和成二之间恰好有一个空位的情况，若我们在空位中再填入一个黑子，就可以得到成四。因此，b 棋型又可称为“冲四”。成和冲都是单独构成的棋型，因此也称为**单线棋型**。单线棋型的另一个重要特征是，两端是否被堵住。比如，图 3-1 中的 a 棋型两端没有被堵住，也称“双活成三”；而图 3-1 中的 c 棋型，尽管也是成三，但一端已经被堵住，成为“单活成三”；而两端都被堵住称为“死成三”。直观上，双活三比单活三、单活三比死三具有更大价值。因此要在估值中考虑这一点。

归纳上述特征，用 cheng[a][b]或 chong[a][b]表示一个单线棋型的估值，这里，a 表示成或冲的棋子数，而 b 表示棋型未被堵住的端点数。比如，图 3-1 中的 b 棋型对应的估值为 chong[4][1] = 50，因为它是冲四，且一端被堵住。在代码中，部分棋型具有 500000 及以上的估值，这表明它们极其重要。比如，cheng[5][0]=10000000，因为这意味着一方已经胜利。又如 cheng[4][2]=500000，这意味着只要 AI 进行理性选择，就很容易达到胜利的局面。

除此之外，先手也会对棋型的估值产生影响。比如，对于 a 棋型，如果当前轮到白方行动，则可很容易地堵住 a，故 a 对于黑方的价值不大。但如果当前轮到黑方行动，则可将其转化为活四，从而达到必胜局面（这种情况也称**准必胜态**）。因此，引入 xianshouCheng[6][3]和 xianshouChong[6][3]两个数组。当先手遇到“活三”、“冲五”等棋型时，将会获得额外的分数。图 3-1 中的 d,e,f 棋型分别显示了一些常见的先手优势棋型。

3.3.1.2 组合棋型的权值

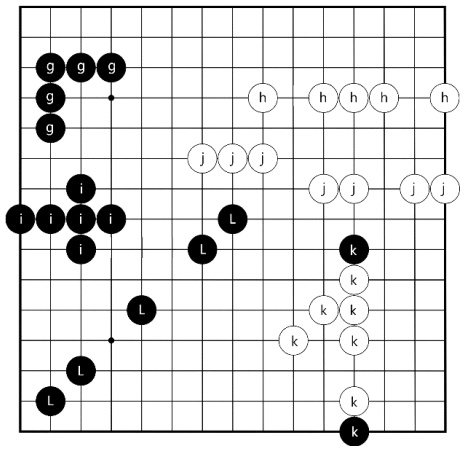

图 3-2 组合棋型举例

组合棋型指的是多个单线棋型的组合，它有时可以产生更好的效果。比如，若黑方有两个准必胜态棋型，那么即便黑方目前是后手，它也已经达到了必胜态（假设白方在下一回合中不会胜利）。因为无论白方对哪个棋型进行防守，黑方都可以在下一回合将另一个准必胜棋型发展为必胜棋型。但是，这样的优势仅凭单线棋型的估值算法是不能体现的。因此要加入对组合棋型的估值函数。

组合棋型的估价函数通过记录局面中每个棋型的个数，按照棋型个数，分析是否有优势组合棋型，并给局面以额外的估值。在图 3-2 中，g~k 棋型展现了五种常见的准必胜态转化为必胜态的组合棋型。L 则展示了一个值得尝试的组合棋型。

3.3.2 单步棋招估值

为了让柱搜索找到最好的若干步行棋，我们需要对当前局面下每一种合法的棋招进行估值，这称为**单步棋招估值**。

单步棋招估值分成三部分：单线棋型、组合棋型、posValue.

该估值算法在对单线棋型与组合棋型的估值上使用了与局面估值相同的估值表。不同之处在于，局面估值需要对全局做出较为准确的评估，耗时较长。而单步估值希望尽量快速地计算每个落子点的估值。因此，单步估值只对指定落子点附近的棋型进行考察，并将该值返回给 createmoves 模块。

单步估值的另一个优化是增加了 posValue 估值表，这个表为每个落子位置分配了一个较小的常数。并且，越靠近棋盘中央的位置获得的分数越大。posValue 可在对局前期，场上落子数量较少的时候，令搜索算法侧重于中央位置的搜索。而一旦对局进行到中后期，posValue 就不再产生实质性的影响。这样，可以避免搜索算法在前期搜索过多的边缘位置，提高了搜索效率。

3.4 Searchmove 模块

3.4.1 算法 1：有截断的 alpha-beta 搜索

我们首先实现了博弈搜索中基本的 Minimax 搜索算法和 alpha-beta 剪枝（以下简称 alpha-beta 搜索），并配合了**截断搜索**技术，即：当搜索深度达到设定的阈值时，停止搜索，直接返回当前局面的评估值。这是由于五子棋搜索终点的深度过深，我们很难在每一步都能直接返回胜负结果。

结合 alpha-beta 搜索+局面评估截断+柱搜索，我们称作**算法** 1。

3.4.2 算法 2：加入散列表

在搜索过程中经常存在不同路径到达同一局面的状况，我们运用 hash 也就是所谓的 **置换表** 或者 **换位表** 技术，目的是消除重复状态的搜索。

在以下**算法** 2 的分析与实现当中，我们设定每一步棋的搜索完成之后 **hash 表清空**。

3.4.2.1 局面 hash 算法：Zobrist 键值

我们使用 Zobrist 键值来给每个局面赋一个几乎不会冲突的 hash 值。

具体地，棋盘上每个位置的每个着棋状态（黑或白）都会被赋给一个随机的 64 位无符号整数的 Zobrist 值。对整个局面进行 hash 时，找到棋盘上的每一个落子，将他们对应的 Zobrist 值异或在一起。此外，若当前局面由白方走，则再异或一个 Zobrist 值 whiteFirstValue；若当前局面由用户走，则再异或上 MinFirstValue。最终结果就是整个棋局的 Zobrist 值。

对于同一盘棋来说，whiteFirstValue 和 MinFirstValue 两个值是冗余的，因为要么白棋是用户，要么白棋不是用户；另外通过观察当前局面中的黑白子个数也可确定是否是白棋先行。

无论如何，我们的 64 位 Zobrist 值都保证了同一个局面有相同的 hash 值，不同的局面几乎不可能冲突。

3.4.2.2 散列表与 alpha-beta 的结合

在 hash 结点中，除了 zobrist 之外，我们还需存储：flag 和 val，分别表示表示估值标签和估值（MINIMAX 值）。flag 的取值有三种： EXACT, LOWER, UPPER ，分别表示估值的搜索结果出来就是 val、至少是 val、至多是 val（后两种情况是因为还未搜完全部分支被 alpha-beta 剪枝了）。

注意到我们假定每次搜索完都清空了 hash 表，那么如果在一次搜索过程中碰到了重复节点，他们的搜索深度是一定相同的（在下面的算法 3 中就不能保证），分情况讨论：

1. flag 是 EXACT，因为搜索深度一样，结果肯定是一样的，直接返回
2. flag 是 LOWER，这只可能发生在 MIN 决策时，而 alpha 在一次搜索中是不降的，那么之前被剪枝了，现在同样会被剪枝，直接返回。
3. flag 是 UPPER，只可能发生在 MAX 决策时，直接返回，理由同上。

由此，可以看出，在算法 2 中，我们引入 hash 表，能够完全消除重复状态，将树搜索转化为图搜索，从而能在有限的决策时间里有效地提升搜索深度。

3.4.3 算法 3：加入迭代加深方法

在**算法** 3 中，我们引入迭代加深方法，且设定每次搜索之后不清空 hash 表。且在 hash 表中中新增成员 depth 和 move，分别表示搜索深度和**已搜索到**的最优行动（注意这不一定是该搜索深度下的最优行动，因为被剪枝）。

在每一步棋的搜索过程中，我们从搜索深度为 1 开始，之后逐渐增大搜索深度重复搜索；浅层的搜索记录的最优行动可以引导、优化当前搜索的**行棋排序**（即搜索顺序），从而提高 alpha-beta 搜索的搜索深度。

具体地，如果遇到一个 hash 表中已经存在的局面：

1. 如果 depth 等于当前所需的搜索深度
    a) 如果是 flag 是 EXACT，直接返回
    b) 如果是 flag 是 LOWER，如果 val < 当前的 alpha，那么就可以直接返回，否则还要重新搜
    c) 如果是 flag 是 UPPER，如果 val > 当前的 beta，那么就可以直接返回，否则还要重新搜
2. 除开以上直接返回的情况，其余情况中，hash 节点中存储的 move，都可以作为**行棋排序**的指导：一种简单的方式是把这个 move 放在第一个扩展。

搜索完一个节点之后，我们需要更新 hash 表。一个简单的策略是：若 hash 表中存储的同一局面的搜索深度不如当前的深度，则覆盖它。

3.4.3.1 Hash 表的清理

由于我们不清空 hash 表，会导致内存占用越来越大：走完一步棋之后，**可能**其他分支中的局面再也不会被搜索到了，而内存却没有被及时释放。

一种清理的方法是在 hash 节点中记录上一次更新（访问）的时间戳 timeStamp。

每隔一段时间，扫描 hash 表，如果很久没被访问过（设定一个阈值），那么就从表中删除它。

3.4.3.2 进一步优化

注意到在**上一步棋**的搜索过程中可能就已经搜索到了当前的局面。

因此我们这一次迭代加深可以不从深度 1 开始搜，而是从 hash 表中已经存储了的深度开始搜。

3.5 扩展功能

3.5.1 悔棋

悔棋指的是分别撤销 AI 和玩家的上一步操作，使得玩家重新下棋的功能。悔棋通过 unMakeMove 函数，将发生的改变分别撤销，以此来恢复之前的状态。

要悔棋，请在玩家下棋时输入“regret”。

3.5.2 复盘

复盘指的是自动记录对局信息，并在下一次程序开始前，询问玩家是否要恢复上次的游戏。如果选择“是”，程序会自动从“record.txt”中读入并模拟行棋记录，然后从上次中断的位置继续开始对局。

要复盘，请确保程序运行目录下已经包含了有效的记录文件“record.txt”，并在程序开始运行时输入“Y”或“y”。

4. 实验结果与评估

实验环境：Windows 10, Visual Studio 2019, 3.1 GHz 双核 Intel Core i5.

为了比较不同搜索技术和优化的效果，我们还实现了一个项目 arena。在该项目中，我们可以指定不同的算法进行对局，并进行对战数据统计。项目中的三个 namespace：AlphaBeta, HashMap, IDSearch 分别实现了算法 1、算法 2、算法 3。为了接下来的叙述清晰不易混淆，**以下使用 AlphaBeta, HashMap, IDSearch 来指代 3.4 节中的三个算法**。注意：这仅是一整个算法的代号，HashMap 中仍然使用了 alpha-beta 剪枝；IDSearch 中仍然使用了 hash 表。

4.1 散列表的优化效果

无禁手五子棋是一种极端非对称的游戏。特别地，黑方有必胜策略。虽然在本次实验中没有通过查表实现必胜策略，但是黑方依然具有极大的优势。

因此，我们无法通过对局 AlphaBeta 和 HashMap 来比较优化效果。实际上，若 AlphaBeta 执黑，则 AlphaBeta 赢；若 HashMap 执黑，则 HashMap 赢。也就是说，我们的搜索优化并没有强到足以推翻黑棋的优势（这也可以从侧面印证我们的算法 1 AlphaBeta 已经是一个较强的算法了）。

并且，在限时 5 秒的情况下，AlphaBeta 和 HashMap 的搜索深度都能达到 9 层，没有显著差异。我们转而比较他们的**等效分支因子**。

在一次 d 层搜索过程中，求解 1+b+b^2+…+b^d=N（搜索节点数）即可得到这次搜索的等效分支因子 b。在一次 HashMap 执黑 AlphaBeta 执白的对局中，每一次决策，我们都重复进行深度为 1~9 的搜索（仅重复执行，相互之间不产生任何影响），将本次对局中所有同深度的搜索的等效分支因子取平均，就得到了这一深度的**平均等效分支因子**。这一过程对应 arena 中的 evaluateHashEffiency 函数。结果如下表：

Table 1. 一次 HashMap 与 AlphaBeta 对局中的平均等效分支因子

深度	AlphaBeta	HashMap
1	8.000	7.971
2	5.139	5.314
3	4.730	4.678
4	4.283	4.127
5	4.069	3.972
6	3.887	3.695
7	3.732	3.571
8	3.659	3.400
9	3.524	3.319

结果分析：可以看到，随着深度的增加，两者的分支因子逐渐减小，这是 alpha-beta 剪枝的效果。在不同深度下，HashMap 的分支因子都小于 AlphaBeta，这说明 hash 表确实起到了减少重复节点的搜索的作用，提升了搜索的效率。

4.2 迭代加深搜索的优化效果

使用迭代加深搜索，在 5 秒的限制内，一般情况下，我们能够将搜索深度提升到 10~12 层（实际深度根据具体局面变化）。

类似上面的情形，我们的 IDSearch 相比 HashMap 并没有强到能推翻黑棋的优势，通过对局胜负不能显著地看出优化效果。

与上面的实验类似，在一次 HashMap 执黑，IDSearch 执白的对局（对应 arena 中的 evaluateIDSearchEfficiency 函数）中，测得的平均等效分支因子如下：

Table 2. 一次 HashMap 与 IDSearch 对局中的平均等效分支因子

深度	HashMap	IDSearch
1	7.968	7.000
2	5.130	4.133
3	4.686	3.627
4	4.137	3.186
5	3.993	3.235
6	3.726	3.158
7	3.589	3.045
8	3.473	2.945
9	3.373	2.859
10	N/A	2.855
11	N/A	2.716
12	N/A	2.430
13	N/A	2.211
14	N/A	2.082
15	N/A	2.017

结果分析：可以看到，在不同深度下，IDSearch 的分支因子都小于 HashMap，这可以验证：迭代加深搜索有利于指导行棋排序，更好地发挥出 alpha-beta 的剪枝作用。

4.3 在线对局

在 Botzone 平台上的无禁子五子棋游戏中，我们的五子棋 AI 可以与其他托管在该平台上的 AI 进行对局，且可以加入采用天梯积分算法进行实力评估的排行榜。

在该平台上，无禁手五子棋游戏 Gomoku 的基本规则是：AI 的运行时间上限在第一轮为 2 秒，此后每轮为 1 秒；且每轮输入结束后，AI 的进程会被中断，下一轮开始时，系统将之前的对局记录作为输入。

因此，在线平台上的 AI 比本次作业中提交的版本要弱一些。这既体现在 AI 的性能上（运行时间长，棋力提高），也表现在搜索效率上（若进程不被中断，搜索中保存的信息可进一步避免重复搜索）。

最终，在该平台上我们的 Bot[2] 截止 4 月 19 日的天梯积分是 1512.37，在排行榜上 156 名 Bot 中排名第 36 名，目前仍在稳定上升中。

这说明我们实现的搜索算法及其优化是有效的，能够得到一个较强的五子棋对弈 AI。

5. 实验收获与体会

这次实验加强了我们对搜索技术的了解，使我们熟悉了常用的搜索优化技术，提高了编程能力。

我们阅读了许多关于五子棋的文献，提高了文献阅读的能力，也对五子棋的规则有了更多的了解。

我们在这次实验中提高了合作与协调能力，对 Github 等软件的使用更加熟练了。

后期对代码的修改、优化、测试加强了我们的调试、编程与沟通能力。

6. 致谢

感谢《人工智能导论》课程的老师和助教提供这次实验的机会和代码框架。

感谢北京大学信息科学技术学院人工智能实验室开发的 Botzone 平台提供了 AI 程序测试与对战的功能。

7. 参考资料与文献

[1] PosValue 的设计：https://blog.csdn.net/zhulong890816/article/details/45459289

[2] Zobrist、置换表、迭代加深等相关资料：http://www.xqbase.com/computer.htm

[3] 徐建. (2016). 五子棋的一种价值的估算. *智能计算机与应用, 006*(005), P.90-92.

[4] 裴博文. (2008). 五子棋人工智能权重估值算法. *电脑编程技巧与维护*(06), 71-77.

[5] 张明亮, 吴俊, & 李凡长. (2012). 五子棋机器博弈系统评估函数的设计. *计算机应用, 32*(07), 1969-1972.

[6] StuartRussell, & PeterNorvig. (2004). *人工智能: 一种现代方法*. 人民邮电出版社.

[2] https://www.botzone.org.cn/user/5e84153d8a935839728237f6；其中 Hope2 是加入排行榜的 Bot。