**基于MIN-MAX搜索的零和博弈AI**

GYH LTT ZZEH ZDY

二O二O年一月

学段汇报信息选题

**摘要**

由于我们对人工智能的现在与未来的价值和功能很感兴趣，我们决定在学段汇报时选择一个与人工智能相关的选题。受到Google DeepMind开发的围棋人工智能AlphaGo的启发，我们决定自己用C++语言编写一个能下五子棋的人工智能。虽然五子棋规则简洁，易于编写游戏主体，却不是零和博弈，即任意局面中五子棋的双方估价和不一定为零。五子棋的这个特性导致其估价函数不易编写。于是我们的指导老师建议我们制作一个能下黑白棋（Reversi）的AI。黑白棋是典型的零和博弈，有利于估价函数的编写与训练。

我们研究与编写的结果是一个和线上的商业黑白棋人工智能下棋时胜率很高的人工智能，附带一个能自己走棋，输出局面，进行落子及吃子操作的头文件。

我们的研究和编写由三阶段组成：

1. 了解黑白棋的规则与套路
2. 编写黑白棋游戏以便套入人工智能
3. 编写人工智能（分为MIN-MAX搜索的编写，估价函数的编制及训练）

经过我们对人工智能的估价函数的训练，我们得到了一个在黑白棋网站上胜率为90%的人工智能。若继续人工智能估价函数的训练，理论上可以达到更高的胜率。

目前我方人工智能为黑时胜率：87.5% 7/8

目前我方人工智能为白时胜率：100% 2/2

关键词：人工智能，黑白棋，MIN-MAX搜索，评估函数，Alpha-Beta剪枝

ABSTRACT

Since we are very interested in the value and function of artificial intelligence now and in the future, we decided to choose a topic related to artificial intelligence during the semester report. Inspired by the Go-playing artificial intelligence AlphaGo developed by Google DeepMind, we decided to write an artificial intelligence capable of playing Gomoku in C++. Although the rules of Gomoku are concise and the Gomoku game is easy to be written, Gomoku is not a zero-sum game, that is, the sum of the two sides of Gomoku in any situation may not be zero. This feature of Gomoku makes its valuation function difficult to be written. So our tutor suggested that we make an artificial intelligence that can play Reversi. Reversi is a typical zero-sum game, which is conductive to the writing and training of the valuation function.

The result of our research and coding is an artificial intelligence with a winning rate of 90% when playing against online Reversi-playing artificial intelligences. Our result also comes with a C header file that can play chess automatically, output the current board, and perform moves properly.

Our research and coding consists of three phases:

1. Get well with the rules and strategy.
2. Code an interactive Reversi game to incorporate the C++ based artificial intelligence.
3. Coding the artificial intelligence (code MIN-MAX search, coding and training of the valuation function)

After our training on the valuation function of our artificial intelligence, we got an artificial intelligence that has a winning rate of 90% on online Reversi websites. If we continue to train the artificial intelligence’s valuation function, we could theoretically achieve a higher winning rate.

The current winning rate of our AI as black: 85.7% 6/7

The current winning rate of our AI as white: 100% 2/2

KEY WORDS: Artificial intelligence, Reversi, MIN-MAX search, valuation function, Alpha-Beta prune

目录

[第一章 引言 5](#_Toc29928896)

[第二章 算法分析 6](#_Toc29928897)

[2.1 评估函数 6](#_Toc29928898)

[2.1.1 稳定子 6](#_Toc29928899)

[2.1.2 地形权值 6](#_Toc29928900)

[2.1.3 行动力 7](#_Toc29928901)

[2.1.4 权值训练 7](#_Toc29928902)

[2.2 MIN-MAX搜索 12](#_Toc29928903)

[2.3 Alpha-Beta剪枝 13](#_Toc29928904)

[2.4限宽搜索 13](#_Toc29928905)

[2.5棋盘哈希 14](#_Toc29928906)

[2.6历史表 15](#_Toc29928907)

[第三章 具体实现 16](#_Toc29928908)

[第四章 总结 18](#_Toc29928909)

# 第一章 引言

黑白棋， 又名翻转棋(Reversi)、奥赛罗棋(Othello)。游戏规则为：共有两位玩家，一人执白，一人执黑，执黑者先走。初始棋盘中央有四枚棋子，二黑二百，摆放位置如图。

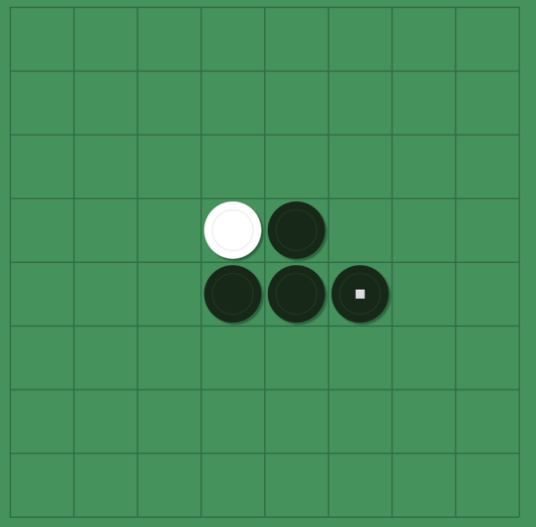


图 1.1 黑白棋的初始状态

Fig 1.1 Initialized status of Reversi

如果某一方的两颗子将对方连续的一些子(方向可以为横，竖及斜)夹在中间，则被夹在中间的对方子将变成己方子。玩家只能将棋子下在能夹住对方棋子并将其变成己方棋子的位置。如图1.2，黑子下F5，与D5的黑子将E5的白棋夹在中间，于是中间的白棋棋子变成了黑棋棋子。需要注意的是，一次落子可以在多方向上夹子。

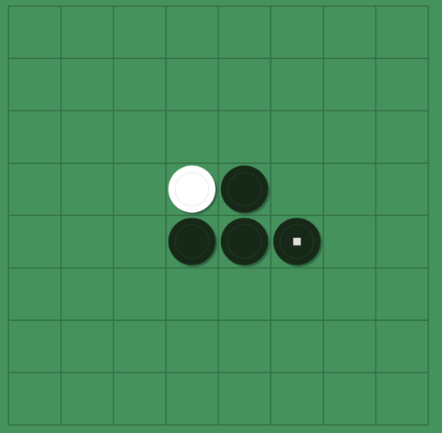


图1.2 黑子下F5后的棋局

Fig 1.2 Board after black puts piece on F5

当我方不能下子时，则对方可以连续落子，直到我方可以下子，反之亦然。当棋局满足以下情况中任意一条时，游戏结束，并以游戏结束时哪方棋子更多判定胜负：

1. 棋盘每一格全被占满
2. 双方均没有落子处
3. 某一方在棋盘上没有棋

# 第二章 算法分析

## 2.1 评估函数

在我们的代码中，某一种情况的价值由稳定子、棋子总权值和行动力决定。具体的决定方式将由训练得出的参数进行判定。

### 2.1.1 稳定子

稳定子即为无论如何都不会被对方吃掉的棋子。例如，则所有与棋盘的某一角相连或与与角相连的棱上的棋子相连的棱上的棋子，完全被对方棋子包围的棋子等。

但是精确的稳定子计算会需要递归多层，甚至计算完整盘棋局，复杂度过高，所以我们只得不精确的计算了目前不可能被被威胁的棋子数量，即8个方向都没有空格[1]。

后文中一个局面的黑方的稳定子数量记为

后文中一个局面的白方的稳定子数量记为

此处board指当前棋局。

### 2.1.2 地形权值

众所周知，在黑白棋中，角落是最好的地方，因为其永远不可能被翻转。同时棋盘边缘也很有优势。于是我们对每个棋子位置赋了权值，如表所示。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 100 | -50 | 20 | 10 | 10 | 20 | -50 | 100 |
| -50 | -75 | 10 | 5 | 5 | 10 | -75 | -50 |
| 20 | 10 | 5 | 2 | 2 | 5 | 10 | 20 |
| 10 | 5 | 2 | 0 | 0 | 2 | 5 | 10 |
| 10 | 5 | 2 | 0 | 0 | 2 | 5 | 10 |
| 20 | 10 | 5 | 2 | 2 | 5 | 10 | 20 |
| -50 | -75 | 10 | 5 | 5 | 10 | -75 | -50 |
| 100 | -50 | 20 | 10 | 10 | 20 | -50 | 100 |

表 2.1 初始估价函数表

Table 2.1 Initialized valuation function table

一方的地势权值即为其所有棋子所在地点的权值之和。

后文中一个局面的黑方的地势记为

后文中一个局面的白方的地势记为

此处board指当前棋局。

### 2.1.3 行动力

即己方有多少种可走的路径。行动力可以影响局面价值是因为，当己方行动力为零时对方可以连走，可以对己方的局面造成极其不利的影响。

后文中一个局面的黑方的行动力记为

后文中一个局面的白方的行动力为

此处board指当前棋局。

### 2.1.4 权值训练

首先设函数表示棋盘状态x的n个局面评分方法中第i个出的结果，本文所述程序的n=3，本文局面评分方法包括稳定子，行动力，地势。

所以黑方在这一局面评分方法中获得的优势就是

设黑方在这一局面评分中获得的优势为。于是对一个棋盘总的评估结果可以看作每个局面评分的加权求和，设相对应的权值为。

我们将一个局面的评估结果表示为下

于是程序实现如下：



即表示黑方在上的优势乘以一个相应的权值，我们要训练的即这个权值。

对于每一个棋盘，我们需要一个训练值（即标准值）用来训练。我们采用该局面进行两层minmax搜索得到的结果作为训练值。

这样我们得到一个训练样例集合，不妨假设我们的训练集合为，这一集合通过程序进行自我对弈生成，对于每一个样例棋盘，其训练值为，我们程序计算出来的估值为。

则定义估价函数对于所有训练样例的误差为：

（为结果美观，在此乘以常数 1/2，对训练结果没有影响）

对于训练方法，我们采用了梯度下降法[2]，即对于一组参数

我们计算出函数在处对于各个参数的偏导数，再往误差函数下降最陡峭的方向调整。

以下为推导：

则此处函数下降最陡峭的方向为其导数矢量的反方向

只需将当前的参数向此方方向前进一小步，即可降低误差，即：

此处为训练步长，为一小值。其值越大，参数逼近最佳组合越快。但步长不可一次设太大，否则有可能错过全局最小值。较好的方法是随着计算减少的值。我们组采用的方式是在训练次数为m时，如果m<=10000，则，如果m<=20000，则，如果m<=50000，则。经过实验，这是较好的一种方法。

由于在一局的不同阶段策略都有可能变化，于是我们给每步走棋都训练了一组w。

实现权值调整的函数如下



自我训练程序的主函数如下：





由于c++系统自带double的范围所限，在此没有对进行求和，而是对其取了平均值。

训练过程中，平均误差的收敛速度随训练次数的增加而减慢，下为其前5000次训练中的平均值收敛过程图表：

图2.1 的平均值收敛过程图

Fig 2.1 The diagram of the process of error convergence

由此可见，刚开始训练时，误差波动较大，但后期逐渐稳定。

前五千次训练过后的平均值达到了5e7，这意味着的平均值小于等于。而实验证明在5000次训练之后虽然误差收敛极慢，但在50000次训练以后测得的平均值已经达到了4100.由于估价函数平均返回值为1e5量级，所以估价函数的误差率仅为4.1%，精度较高。

## 2.2 MIN-MAX搜索

定义搜索深度depth，其在棋局中的意义为在现局面搜索往后的depth回合。当递归到第depth层时停止递归。若本回合自己走，则称此回合在搜索树中的节点为MAX节点，否则此节点即为MIN节点。当前节点为MAX节点时，搜索最利于自己的情况；当前节点为MIN节点时，搜索最不利于自己的情况，并将权值（MAX/MIN）传递给其子节点节点。最后递归到深度depth时，选择总价值最大的情况进行落子。

例如图2.1，搜索深度为3层，两个min节点分别选择自己子节点里估值最小的节点，而max节点则选择它的子节点，即两个min节点中估值最大的节点。

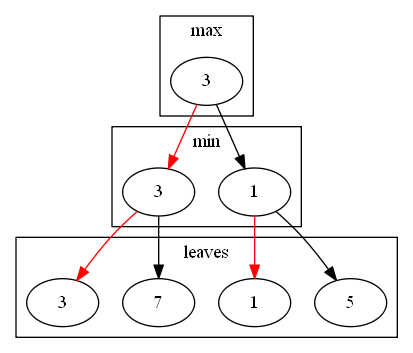


图2.2 MIN-MAX搜索的示意图

Fig 2.2 the schematic diagram of MIN-MAX search

## 2.3 Alpha-Beta剪枝

Alpha-Beta剪枝即：在搜索过程中，为局面的价值设定上界与下界。若某种局面的价值小于下界，停止搜索此情况；若价值大于最大值，将最大值替换为此价值，继续搜索；若价值在最大最小值之间，继续搜索。

例如图2.2，当搜索到右边的min节点并搜完它左边的叶节点时，发现alpha小于beta，即可退出搜索，不用搜索右边的叶节点。

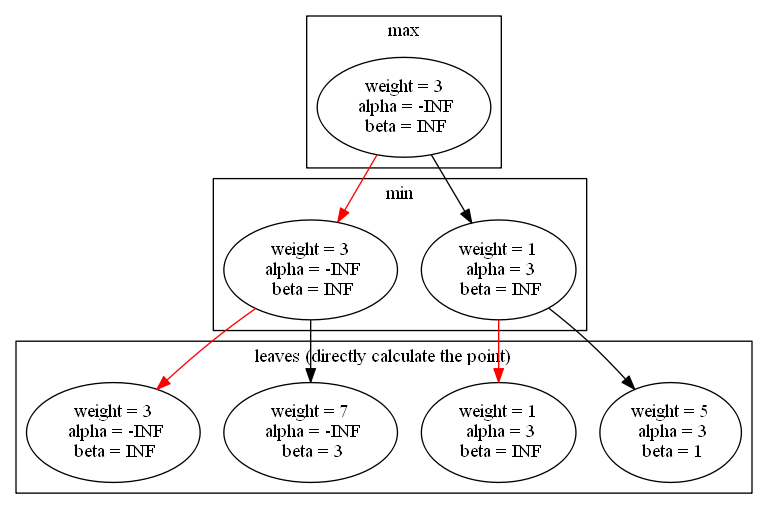


图2.3 包括α-β剪枝数值的MIN-MAX搜索示意图

Fig 2.3 the schematic diagram of MIN-MAX search that includes the value of α-β prune

## 2.4限宽搜索

当需要搜索的节点过多时，搜索时间就会上升，导致效率变低。我们可以按照每个节点的估价值，限制每一层的可以向下搜索的节点数。

例如图2.3，由于子节点太多，可以只搜索估值前7大的子节点。

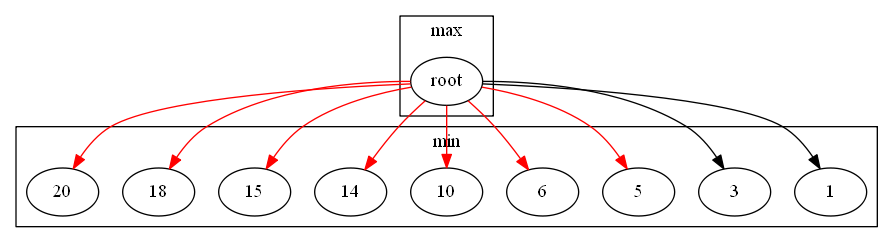


图2.4 限宽搜索

Fig 2.4 width limited search

## 2.5棋盘哈希

为了方便记录棋盘状态，我们需要将棋盘压缩成一个数，这里采用了Zobrist hash算法，该算法将棋盘上的每一个位置赋值为一个大随机数。某一方的哈希值就是其占领的所有格子的异或和。

实际计算中，可以每次根据上次的棋盘直接推出下一个棋盘的哈希，节省时间。每次下一颗子，就将哈希异或这个格子的随机数，失去一个子，就再异或这个随机数一下。我们的程序通过将黑方和白方的32位哈希合并成一个64位的哈希来得到整个棋盘的哈希值，以下是代码实现：

## 

## 2.6历史表

对于搜索过的棋局，可以根据其搜索结果，下次再搜到该棋局时，优先考虑上一次搜到的最优情况。这样就能增大Alpha-Beta剪枝的次数，优化运行时间。

于是我们采用Zobrist hash算法计算出棋盘的哈希值，并将这一值用作存储棋盘搜索结果的键。以下是代码实现：



# 第三章 具体实现

在实现过程中，我们定义了头文件board.h并在核心代码中引用以方便代码书写。

board.h头文件可以允许你在程序中定义类型为Reversi\_Board的结构体，其中含有基础估价函数，基本的棋盘，可以直接进行吃子操作的落子函数，输出棋盘函数，以及可以判断某方在当前棋局的某坐标可不可以进行落子的函数等等。

在核心代码中，我们通过有Alpha-Beta剪枝的MIN-MAX搜索，实现了依据对对手未来六步的走法的预判下棋的程序。这个程序在搜索的深度为6或7时的反应时间仍没有超过1秒，性能较强。限于篇幅，在此不展示代码，读者可以访问<https://github.com/bdfzoier/Reversi>自行阅读，此处大概叙述程序框架（主要包括定义的对象及其方法）：





# 第四章 总结

我们自己编写的机器人打败了coolmathgames.com，4399.com，微信小程序等多个AI以及许多真人玩家。这是因为我们采用的方法并非简单的Min-Max搜索加上α-β剪枝，我们的方法中还包括有对估价函数的训练。经过训练的估价函数并非死板的只会计算的机器，而是会通过局面的各种参数并加深搜索来判定该走哪一步棋的AI。看着AI一步一步将对手的棋子占为己有并将对手击败，我们 都很开心。希望我们的机器人以后能再进一步的优化后更加强大，击败更多更强的对手。

# 致谢

首先感谢北大附中信息中心的信息导师XR，您为我们提供了走向信息学的大门，为我们开设了信息学科，并为我们整个学段汇报的研究方向打下了扎实的基础，同时在研究的过程中对我们严格要求。

还要感谢我们的父母，在我们学习的过程中为我们提供了尽力的支持。

# 参考文献

1. 黑白棋AI：局面评估+AlphaBeta剪枝预搜索 [2018-05-09] https://zhuanlan.zhihu.com/p/35121997
2. 《机器学习》(美)Tom Mitchell
3. 柏爱俊 2007 年 10 月 几种智能算法在黑白棋程序中的应用