经典五子棋游戏中对弈策略的算法设计及评估

陈柘宇*[1], 濮成风[1], 陈树伦[1], 孙若凡[2], 吴之琛[2] , 曹文[3]

[1]*2019*级*10*班, 江苏省常州高级中学

[2]*2019*级*11*班, 江苏省常州高级中学

[3]指导教师, 江苏省常州高级中学

2021.3

修订 2021.9

摘要

我们针对经典五子棋游戏，主要围绕启发式对抗搜索，提出并实现了若干算法并对其分别进行了评估。为方便进行测试，实现了一个对战测试平台。在对战测试研究的基础上，我们综合各算法并总结思路提出一个具备拓展性的完全信息静态博弈算法的设计思路。通过对算法的分析，试验性地提出一项衡量棋类问题搜索算法效果的评价指标——搜索有效层数并给出了初步的讨论。并尝试使用该概念解释探究的结论并指导算法的设计。

*联系邮箱:cipsum@163.com

1 背景

从20世纪计算机发明以来，人们利用计算机作为工具对“智能”的探索就从未停止。无论是上个世纪风行一时的专家系统还是近年来的学术界和工业界的热门深度学习，都是人们利用计算机优于人脑的先天优势——或是极强的存储能力，或是超人的计算能力——在某一问题上完成仅凭人脑难以企及的成果。而随着计算机技术，尤其是硬件技术的飞速发展，在可预见的未来利用计算机模拟的智能将会越来越强大。这已经隐隐地成为了时代的大势所趋。

五子棋游戏作为国际上流行的智力竞技项目，历史上有很多关于它的研究成果。很早就有人注意到五子棋游戏中先手具有极大的优势，为此，人们提出了禁手的概念。不过已有论文表示无论是否存在禁手都可证明先手必胜[1][2][3]。人们在此基础上提出了大量改进规则，有“交换”思维（一方指定开局，另一方选定自己的颜色）、“泡沫”原理等。但是我们因为想要探索智能、搜索与棋类游戏，而不仅是五子棋，因此我们还是采用最基础的规则进行研究。

我们发现五子棋推演的基本规则和搜索的一般思路极为相似。这也暗示了用搜索解决五子棋的可行性。

我们探究五子棋问题的对弈策略的探究过程、改进思路也对其他更具现实意义博弈问题的简单解决方案设计开发具有启发意义。

我们在做IOI(International Olympiad in Informatics)2020国家集训队作业题的时候做到了一道五子棋相关的问题，受此启发，针对一些较为简单的算法进行了探究。

2 研究前期准备

2.1 五子棋相关背景信息

棋盘 棋盘由横纵各15条等距离，垂直交叉的平行线构成，在棋盘上，横纵线交叉形成了225个交叉点为对弈时的落子点。

先手优势 指在五子棋游戏中，因为绝大多数的规则下先手都有更大的胜率，所以我们称先手具有优势。我们将这个优势叫做先手优势。

基本战术 我们发现这是一些对于AI设计具有启发性的战术思维。

- VCT(Victory by Continuous Threat) 通过连续不断的冲四、活三、做杀等先手进攻取得胜利。
- VCF(Victory of Continuous Four) 以连续不断的冲四取得胜利。又称“追四”。

- 风车轮指在一定区域内来回地绕着连续进攻的战术或技巧。又称“风车胜”。

2.1.1 已有棋类AI设计思路

AlphaGoZero 核心思想是利用蒙特卡洛搜索树，在局面和动作之间建立联系，对局面进行胜率评估。通过改进的深度学习的方法进行迭代更新。在决策初期引入了随机化。参见AlphaGoZero原理示意图。该方法对我们的启发意义在于用于迭代的模型可以是搜索树的形态也可以是决策时使用的估价函数。

DeepBlue 核心思想是进行大规模搜索以及借助人类已有棋谱进行方案评估。对于我们的参考意义在于可以针对已有棋谱建立数据库并根据数据库来进行搜索决策。不过经过讨论后我们认为这个方案对于我们来说略显难以实行——主要由于已有棋谱的合理利用对于我们来说需要花费太多时间，因此决定暂且搁置。不过因为五子棋具有更为重要的一些局部子结构，所以对于一些简单情况进行特殊判断是一件必须要做的的事情。

2.1.2 研究目标

研究性学习 因本课题为学校研究性学习课程的一部分，故我们将在研究过程中力求独立、科学地完成相关探究。

构建五子棋对战平台 [1]我们在前期并没有找到适合我们研究需要的平台，所以我们自己构建了一个。[2]

实现算法并分析 [3]我们将就较为简单的估价函数、对抗搜索进行探究，并适当利用遗传算法进行研究。在实际测试结果的基础上，试着去构建理论对探究结果进行解释，并进一步给出在类似问题上具有启发意义的算法设计思路。

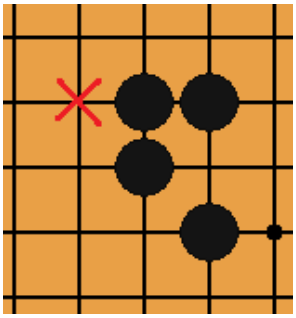
[1]该方面主要开发人员为陈柘宇、濮成风、吴之琛，陈树伦与孙若凡负责测试
[2]见附件：程序框架说明文档、对战平台使用指南
[3]主要负责人员为陈柘宇、孙若凡、陈树伦，所有成员均有贡献

3 下棋程序设计与实现

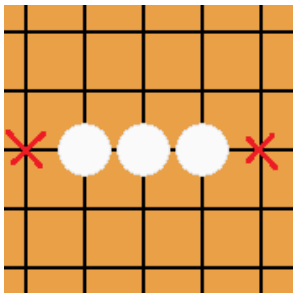
3.1 具体算法设计

3.1.1 AI2——估价函数

我们知道，在经典的五子棋游戏中，如果出现了固定的形状，我们更倾向于选择能够让我们获胜的局面。例如，目前黑手执棋，棋盘上出现了二二的情况（无禁手），如果黑手其连成三三，那么就进入了一个必胜的局面(见图:AI2必胜局面)。再例如，目前黑手执棋，白方已经出现了一个三连，如果黑手不去堵住它，那么必然会进入必败的局面(见图:AI2必败局面)。对于一个熟练五子棋的玩家而言，这种局面是相当容易看出的。但目前计算机只能进行一些数值的计算，无法像人脑进行复杂的联想。通常，我们会引用三种方法，第一是引入估价函数的概念，第二是进行对抗搜索，第三是将两者融合。在这个算法中，我们将使用估价函数。



(a) AI2必胜局面



(b) AI2必败局面

估价函数，顾名思义，它是对当前局面是否对己方有益的定量估计。它的输入是一个棋盘，或者是棋盘的一部分，它的输出通常是一个数值或者一个落子的坐标。例如，我们可以设计一个估价函数，让黑方能够捕捉到已经连成三连的情况，那么我们可以这样设计：

1. 输入一个局面。

2. 遍历所有连成三连的黑子，如果它的两边都是空的，那么在这两个空位上加上100分。

3. 找到分数最大的空位，若找不到，那么是平局；否则返回这个空位的坐标。

比如，将某个局面输入到这个函数中，我们能得到如图所示(见图:AI2分数示例)的分数。接着，我们会找到分数最大(200)的空位，得到它的坐标。

不过这只是一个演示，为了得到一个比较好的估价函数我们通常会经行精心地设计，它得权衡

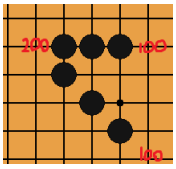


图 1: AI2分数示例

进攻与防守这两个模式。为了进行研究，我们选取了如下估价函数:

$$f_{x,y} = \sum_{s \in S(x,y)} g(s)$$

$$g(s) = \begin{cases} 50^{2n(s)-1}, & n(s) > 0 \\ rand(0, 2499), & n(s) = 0 \\ -50^{2n(s)}, & n(s) < 0 \end{cases}$$

估价函数大体为对每五个合法的连续位置进行扫描计算。其中S表示所有包含(x, y)的合法的五个连续的位置的集合。$n(s)$表示一个五个连续的位置中自己的子个数-对方的子个数。若这五个位置中既有自己的子又有对方的，则$n(s) = 0$。[4]

虽然面对这种估价函数的时候对手存在一种必胜的策略，但它的想法具有启发意义，远超完全随机的AI1。在下面，我们将广泛运用这个思想。

3.1.2 AI4

基于维护估价函数的贪心，引入随机数以增加不确定性并避免局部最优解以及估价函数的局限。估价函数基本模型为：

$$f_{Board} = \sum_{s \in S} g(s)$$

$$g(s) = \begin{cases} a_1 2500^{n(s)-1}, & n(s) \geq 3 \\ a_2 2500^{n(s)-1}, & 0 < n(s) \leq 2 \\ rand(0, 2499), & n(s) = 0 \\ -a_3 2500^{n(s)-1}, & n(s) < 0 \end{cases}$$

其中a_1, a_2, a_3为不同的权值，S为所有合法的五个连续位置的集合，$Board$为需要评估的局面。

[4]来源：2020信息学奥林匹克竞赛国家集训队作业

在这里$a_3 = 100, a_2 = 10, a_1 = 1$。可以发现这个估价函数和AI2的设计思路基本上是一样的。差别在于针对活三活四的情况进一步细化了判断函数，可以直接利用函数判断是否有即将必败局面的出现，而不用去特判。但是该方法的局限性在于只能判断一步之内必败。

3.1.3 AI5——改进估价函数

在AI4的基础上改进了估价函数.

我们在测试的时候发现,AI4在和人类对战的时候会在一些必胜/必败局面上判断错误,因此我们加入了判断两步内必胜的特判,增强AI的强度,实际测试效果比AI4略强一些但是没有非常大的优势.

两步内必胜的特判有三个连着并且没有被阻挡的子,四个连着的子.

AI5在AI4的基础上同时对估价函数的计算进行了一些优化,方法大体为事先扫描好连续的五个格子,然后用一个数组记录这些连着的格子的情况.

3.1.4 AI6

由于AI2的估价函数存在针对它的必胜策略，这个AI通过将不同位置处的权值设置为不同的值来处理这个问题。

故采用的如下的位置权重函数：

$$w(i,j) = 50 + \min(i, n-i-1) + \min(j, n-j-1)$$

其意义为 50 加上点 (i,j) 到棋盘四条边的最小距离。

估价函数为

$$f_{Board} = \sum_{s \in S} g(s)$$

$$g(s) = \begin{cases} \prod_{(i,j)\in s} w(i,j)[(i,j)\text{上有棋子}], & \text{S 个位置中有且仅有己方棋子,} \\ -20\prod_{(i,j)\in s} w(i,j)[(i,j)\text{上有棋子}], & \text{S 个位置中有且仅有对方棋子,} \\ 0, & \text{其他情况} \end{cases}$$

这个AI引入的对抗搜索的初步思想，它会考虑对手对己方落子点的回应。

即设我方落子点为 (x,y) 它将最大化

$$\min_{(x',y')} f_{Board+(x,y)_1+(x',y')_2}$$

其中 $Board + (x,y)_1 + (x',y')_2$ 代表在 $Board$ 的基础上添加 (x,y) 处的己方棋子与 (x',y') 处的对方棋子得到的局面。

这里并没有采用任何可能加速枚举过程的方法，因此时间复杂度总为 $O(n^4)$，与棋盘状态无关。

3.1.5 AI7——改进估价函数

深入研究了AI4的估价函数的判断逻辑，并针对其本质逻辑简化了计算方式。主要是不在通过手动写循环枚举每一种五连子的情况，而是进行自动枚举来简化后续调整所需工作量。

估价函数为：

$$g(s)=\begin{cases} rand(1,Num_5), & n(s)=-6 \\ rand(1,Num_6), & n(s)=0 \\ t(s)Num_1Num_2^{-n(s)-1}, & -5\le n(s)\le -1 \\ -t(s)Num_1Num_3Num_2^{n(s)-1}, & 1\le n(s)\le 2 \\ -t(s)Num_1Num_4Num_2^{n(s)-1}, & 3\le n(s)\le 5 \end{cases}$$

这个函数和AI4的估价函数本质相同。

其中设置的$Num_1=30, Num_2=900, Num_3=40, Num_4=90, Num_5=799, Num_6=10$。

注意到其中$n(s)$的含义做了更改：对于5个位置，若为空，$n(s)=0$，若全为自己的棋子，$n(s)$等于负的自己的棋子个数，若全为对方的棋子，$n(s)$等于对方的棋子个数，否则$n(s)=-6$。

$t(s)$为其中连续同色段的段长的平方和。

3.1.6 AI8——遗传算法

为AI7加入了遗传算法，希望自动优化其估价函数的参数。对于该算法的详细分析下文中有。

3.1.7 AI10——对抗搜索

该 AI 的估价函数定义较为复杂，具体为

$$g(s)=w(n(s))$$

$$w(n)=-\infty[n=-5]-10^{40}[n=-4]-2\times10^6[n=-3]-49990[n=-2]-100[n=-1]$$
$$+10[n=1]+20000[n=2]+50000[n=3]+9\times10^5[n=4]+\infty[n=5]$$

$$f_{Board}=\sum_{s\in S}g(s)+[mC>0]2\times10^7+[mC>1]10^{10}-[oC>0]10^{13}-[oC>1]10^{25}+10^{30}c_1-10^{50}c_2$$

其中 c_1 表示己方两侧均为空的四连数量，c_2 表示对方两侧均为空的四连数量，mC 表示己方能够在一步内能够获胜或使 c_1 增大的结构数量，oC 表示对方满足类似条件的结构数量。

该AI保留了对抗搜索，但只考虑看起来较优的情况（即离已有棋子较近且估价函数较大的情况）来增大搜索层数。

具体搜索规则为：

（1）定义当前局面的最小覆盖矩形是能覆盖所有棋子且与边界平行的最小矩形，每次扫描矩形内的空位以及矩形外圈一层的空位，并根据当前层数保留较优的一定数目的方案。

（2）枚举这些方案，同时递归考虑对方的决策，当递归层数达到5时，停止搜索并用此时的权值返回。

（3）采用 Alpha-Beta 剪枝，这将在后面进行介绍。

主体部分复杂度 $O(Kn^2)$，其中 k 是与搜索规模限制有关的深度。

3.1.8 AI11——对抗搜索

该AI是基于AI10的改进。

把 $w(n)$ 在 $n=4$ 处的值修改为 -10100；把 $n=-2$ 处的值修改为 -29990（在 $n=4$ 是改成负值是为了防止在搜索过程中过度强化最终状态冲四的重要性）。

对 f_{Board} 加入了一个不超过1的随机量扰动。

放宽了搜索层数限制以及每层的搜索数目，并引入迭代加深策略，同时增加了单步总计10s的搜索时间限制。

迭代加深策略：从5层限制开始搜索，每次搜索结束后保留最优的 70% 的决策，并将层数限制加 2 开始下一轮搜索。在13层下完成搜索后将停止搜索并在此时的最优点落子。

除去某些较为复杂的局面外，大部分的正常局面都可以在10s的限制内完成或将近完成搜索。

3.1.9 AI10与AI11用到的Alpha-Beta剪枝

这里介绍AI10与AI11用到的Alpha-Beta剪枝策略，它的作用是及时排除无效决策，减少耗时。

当 AI 在搜索状态 P 时，以下一步为先手为例，AI 将枚举先手在搜索规模限制下的决策 $A_1, A_2, \cdots, A_k$，若 AI 已完成的至少一种决策的考虑，记已发现的最优决策产生的权值为 α，当在下一个决策 A_j 中发现对手的某一策略将导致最终权值劣于 α 时，那么决策 A_j 一定是不优的，从而可以跳过对 A_j 其余情况的搜索。

一个较为极端的例子：如果我方的某一步 (x,y) 将给对方留下一个四，那么发现对方可以立刻获胜后将不必继续考虑己方落在 (x,y) 时对方的其他应对，因为这一步是必败的。

上面的分析过程同样适用于对对方落子点的优化。

3.1.10 AI12——对抗搜索

该算法在程序中的名字为EntityPCF。

该算法的基本思路与AI10大致相同，因此不再展开介绍。

其中比较特别的是我们的估价函数使用的是AI2的改进版，估价函数的效果预计会优于AI10的函数。我们利用遗传算法优化了他的参数之后成功使其在先手对局的情况下可以做到面对AI10不会落败。这已经是一个巨大的进步了。

其中的搜索是最正统的对抗搜索，并没有进行剪枝，因此运行效率极为低下。而这也限制其搜索的状态空间大小不超过10^5种。大致可以做到搜7层，每层搜6个最优解。每层会使用$O(n^2)$的时间对局面的每个位置进行估价并选择其中价值最大的位置。而搜到末尾整个状态的估价则用整个局面所有位置估价函数之和。

3.1.11 AI13——对抗搜索

该算法在程序中的名字为EntityCzy。

起因是我们发现了AI12的估价函数的一个硬伤：我们注意到在对抗搜索的时候我们应该尽量去往对方想下的位置落子，此时我们应该为对方的活三之类的局面赋很大的权值，但是在末尾评估的却应该为对方的必杀局面赋极小的权值。这个观察启示着我们两部分的估价函数应该有不同的处理，否则你要么会浪费大量计算力在没有意义的局面上，要么会对终局产生错误的评估。

因此我们在AI8的基础上增加了对抗搜索的部分。因为AI8的函数是对局面的评估，所以如果直接使用会导致单次估价的复杂度上升到$O(n^4)$，当然可以做到$O(n^2)$，但是常数问题同样不可接受。所以我们将$g(s)$的值加到了该次s每个位置上面，以实现对于每个位置的简单的$O(n^2)$评估。

而对于AI12的硬伤的规避，我们对抗搜索决定下一层搜索对象的时候使用的$g'(s) = |g(s)|$。同时为了避免出现自己搜出来必败就直接放弃的情况，我们将一个局面的权值定义成一个二元组$v = (val, depth)$，其中val表示权值，若必胜为inf，若必败为-inf。我们希望必胜的层数尽可能浅，必败的层数尽可能深。根据该逻辑我们设计了v上面的偏序关系。

3.2 算法关键设计细节解释

AI4为代表的估价函数 这个估价函数的基本思路是对于棋盘上所有连续5个位置进行计算，基本形式为如果这五个位置存在两种颜色，就不管；如果只有单个颜色，对局面权值的贡献是关于五个位置中棋子个数的指数函数。这件事情保证了活三之类的情况一定会被判断出来并且处理掉。但是他的问题在于连续五个的判断不能进行更长远的布局，整个思考是直线形的，所以比较适合放在对抗搜索的末层，来等效地扩大搜索层数。不过AI4,AI5,AI6,AI7的问题在于他们由于我们时间安排的关系，都没有加入对抗搜索。

AI10为代表的对抗搜索 这个对抗搜索是通过控制每层只搜索最有希望的几个来实现在合理的时间内搜索出一个较好的解，详见AI10与AI11的算法详细讲解。因为其搜索的层数在普通计算机上10s以内基本可以达到11层，而最差也会有7层，所以可以保证普通的必死局面一定会被判断出来。

AI8为代表的遗传迭代 这其实是现代机器学习的最基本的思路：通过迭代来调整模型参数。我们针对这个问题和我们的能力，选用了代码逻辑简单但是效果不错的遗传算法。我们发现一局对局平均需要使用0.8s的时间，因此我们决定每轮迭代保留最优解、次优解并根据最优解随机产生3个新的模型。每个模型统计50局对局的终局评估函数（不同于算法中使用的估价函数，是根据如果输了，拖得越久越好；如果赢了，越快获胜越好设计的）并将其作为该模型的估价。因为在实际操作时发现执白与执黑参数演化方向并不一样，所以我们对其做了区分。

4 程序的比较与分析

4.1 空间复杂度分析

4.1.1 理论空间复杂度分析

我们利用数学方法，分析出了各个AI的理论空间复杂度。见表1

4.1.2 实际空间复杂度分析

由于时间仓促，并未来得及完成全部的分析。仅就目前的结果来看，我们能确定的是每一个AI在N=15的时候所用的内存空间均不超过1GB。

4.2 时间复杂度分析

4.2.1 理论时间复杂度分析

我们利用数学方法，分析出了各个AI的理论空间复杂度。见表1

4.2.2 实际时间复杂度分析

由于时间仓促，并未来得及完成全部的分析。仅就目前的结果来看，除了AI10、AI11、AI12和AI13以外，所有的AI在所有情况下均能在0.5s内下完一步棋。AI10在前期10个子以内能达到类似的效率，后期效率随棋盘复杂度上升而逐渐下降。AI11与AI10类似。但AI11中加入时间限制机制，保证在10s以内给出一个解。AI12和AI13严格计算了搜索空间大小，能保证时间不超过8s。

AI编号	AI基本原理	AI理论空间复杂度	AI理论时间复杂度
AI2	基于计算估价函数的贪心	$O(N^2)$	$O(CN^2), C=5$
AI4	基于维护估价函数的贪心	$O(N^2)$	$O(N^4)$
AI5	改进估价函数	$O(N^2+T)$	$O(CN^2), C=8$
AI6	改进估价函数	$O(N^2)$	$O(N^4)$
AI7	简化AI4估价函数	$O(N^2)$	$O(N^4)$
AI8	在AI7的基础上加入遗传算法迭代	$O(N^2)$	$O(N^4)$
AI10	基于估价函数的对抗搜索	$O(KN^2)$	$O(KN^2)$
AI11	基于AI11利用迭代加深的优化	N/A	N/A
AI12	基于AI2利用对抗搜索的优化	N/A	N/A
AI13	基于AI8利用对抗搜索的优化	N/A	N/A
注：N为棋盘大小；K为搜索的情况数；T棋盘上五子连线的总情况数			

表 1: 时空复杂度分析

编号	AI4	AI5	AI6	AI7	AI8	AI10	AI11
AI4	50.7%, 48.1%	62.4%, 36.6%	81.8%, 17.5%	65.4%, 33.6%	59.8%, 39.5%	3.4%, 96.6%	6.0%, 94.0%
AI5	40.2%, 57.8%	49.0%, 50.1%	34.2%, 55.6%	59.3%, 38.1%	51.0%, 47.0%	25.2%, 65.5%	0%, 100%
AI6	68.4%, 22.8%	68.6%, 31.4%	N/A，先手	67.2%, 36.3%	87.1%, 10.8%	N/A，后手	N/A，后手
AI7	54.2%, 42.1%	63.0%, 35.6%	63.5%, 36.5%	78.1%, 19.5%	55.5%, 43.3%	99.0%, 1.0%	96%, 4%
AI8	55.4%, 40.3%	60.6%, 37.8%	76.0%, 24.0%	84.3%, 14.4%	56.6%, 42.7%	96.6%, 3.3%	100%, 0%
AI10	99.9%, 0.1%	100.0%, 0.0%	N/A，先手	100.0%, 0.0%	100.0%, 0.0%	N/A，先手	N/A，后手
AI11	99.5%, 0.5%	100%, 0%	N/A，先手	100%, 0%	99%, 1%	N/A，先手	N/A，先手
其中N/A表示确定性AI重复对弈的结果完全相同。 列为执黑者，行为执白者，每一格均是在标准五子棋（不含禁手）的情况下对弈1000局的结果。 括号左侧为黑方胜率，右侧为白方胜率							

表 2: 算法强度比较

4.3 算法强度比较

在所有AI中，除去Entity1为随机落子，Entity2胜率过低（但以100.0%的胜率执白击败它并不简单）与Entity3为从终端读入外，我们测量了以下对阵情况，其中Entity4与5为随机性算法，Entity6与10为确定性算法（即相同局面给出的下法不变）：(部分AI对战结果见表2)

表2中，黑白胜率和不为100%是因为可能会填满棋盘后仍无法分出胜负。

总体来看，黑方优势较白方优势明显。5个AI实力关系约为$AI4 \approx AI5 < AI6 < AI10 < AI11$，人机对战记录也支持了这一关系，但数据不足，不便给出。

我们暂时不能确定AI4和AI5在自我对弈中先手优势不明显的原因。

上表也说明随机化算法未必比确定性算法好，这一点与我们的估计并不吻合。

4.4 进一步分析

4.4.1 AI8的分析

AI8是利用遗传算法来优化AI7的估价函数的参数。原始参数是经过简单分析设计的。经过迭代之后的确可以达到不错的效果：胜率可以上升5至10个百分点。不过因为估价函数本身的局限性，他依旧无法抗衡加入对抗搜索的算法。不过从胜率表可以看出他确实比AI7本身更为优秀。

估价函数具体各项数值参见表3。我们发现其实若干参数的变化仍没有明确收敛的迹象，只是变化速度在降低，但是现在进行分析已经足以说明其变化趋势并得出一定结论了。

角色	迭代轮数	Num_1	Num_2	Num_3	Num_4	Num_5	Num_6
原始	0	30	900	40	90	799	10
执黑	40	29.153065	900.865529	34.812081	118.971993	689.591161	10.554894
执白	40	30.168348	897.030364	29.570714	86.135963	1336.091075	11.988389
执黑	120	30.945376	903.967744	28.987532	38.239987	324.778935	13.718610
执白	120	27.753161	889.600012	5.938275	207.790961	994.864699	5.680720

表 3: 估价函数迭代结果

针对其数值变化进行分析，我们可以发现：Num_1, Num_2没有本质变化，这是合理的，因为指数函数的底数的变化会带来大量额外变量的变化，而我们的各项参数变化每次的步长约为相对0.5%；Num_3有小规模减小趋势并基本收敛，可以初步认为不能短时形成攻势的位置在该估价函数下没有那么重要；执黑时Num_3, Num_4相对于Num_5上升，可以认为是在降低随机的影响，而执白时恰恰相反，这说明在这个估价函数下执白需要一定的随机性才有可能获得较好的结果。

我们认为这个函数的变化趋势对于改进设计以及更好地调整迭代的参数有极其重要的指导意义。对于将估价函数与对抗搜索结合也有指导意义。他也启示我们遗传算法的效果和估价函数本身有极大的关系。

4.4.2 AI12和AI13的分析

由于时间原因，我们没有进行大规模统计。

仅就有限的数据来看，AI12基本可以战胜没有加上对抗搜索的AI2至AI8，其中面对AI2胜率较高，这与AI2和AI12利用的估价函数极为相似有关。可以在先手的情况下以约40%的胜率战胜AI10，后手面对AI10以及先后手面对AI11胜率极小（小于10%）。一部分是由于其搜索层数极少，另一部分是我们在上文中介绍过的AI12的估价函数的硬伤导致他出现了本不应出现的判断失误。

AI13可以在面对不加入对抗搜索的对手的时候基本可以做到不败。而面对AI10时，先手有约50%的胜率，10%的败率，后手胜负情况基本相反；面对AI11的时候，几乎总是平局。这种情况出现的原因据判断为它假定了一个极具进攻性的对手，所以策略多具有短期的保守性。但是因为搜索方式的限制导致搜索的层数并不能达到AI11的级别，更妄谈进行战略布局了。

AI12和AI13与AI10和AI11的对比，一方面告诉我们估价函数在这个问题里有很重要的地位：如果利用估价函数对局面进行了合理的估计，搜索的实际有效层数会比较高；另一方面告诉我们搜索

方式本身在这个问题里有很重要的地位，因为合理的搜索方式可以尽可能地去探索估价函数给出的搜索空间中有价值的部分，同样可以提升搜索的有效层数。

综合AI8的分析，我们发现遗传算法在该问题中的意义在于人类可能并不能很好地分析在搜索中一个函数中的数值到底意味着什么以致于只能给出极其粗略的参数设计。遗传算法可以在实战中“总结”函数参数的“实际”意义，尽最大可能发挥该函数的威力。

4.4.3 其他观察

观察1 我们发现在开局阶段，程序常出现无法理喻的失误，至6手以后会好很多，至20手后，常能完成极其精妙的进攻与防守。

观察2 设计得不规范的对抗搜索，会在自己认为自己必输之后自暴自弃，导致确实输掉了。然而在这种时候，很有可能还有一线生机或者该必输局面并不会为对手所发现。

观察3 在绝大多数情况下，算法缺乏战略意识；或者说算法的战略意识仅为尽量争取每一场小胜利以换取大胜利。在类似五子棋的情景中，该意识问题不是很大，但是在类似象棋和围棋的情境下就绝对不行了。

观察4 通过调整参数，可以使算法在下棋风格上表现出极强“进攻欲”。一个具体的方法是使用较强的算法（AI13）作为黑方与较弱的算法（AI4）跑遗传算法，20代之后再与AI13对弈，可以很明显地感觉到其攻击更加急促。具体来说，我们观察到面对类似的子结构的时候，同一算法在落子位置上给人类似的感觉。

5 理论构建与分析

5.1 搜索有效层数

根据AI10,AI11,AI12,AI13的分析，我们针对类似的算法模型提出一个具有通用性的算法评估概念：搜索有效层数。为给出该概念的定义，我们先给出一些相关定义与猜想。

定义 5.1 *先后手胜率 我们记算法 A_1 对算法 A_2 的先手胜率为 $pre(A_1, A_2)$ ，后手胜率为 $suf(A_1, A_2)$ ，综合胜率为 $com(A_1, A_2) = pre(A_1, A_2) + suf(A_1, A_2)$ 。*

定义 5.2 *搜索算法等效 我们称两个该问题下的搜索算法 A_1 与 A_2 等效，当且仅当他们剔除策略倾向的影响后在统计意义下 $com(A_1, A_2) = 1$，记作 $A_1 \approx A_2$ 。类似的，我们称 A_1 弱于 A_2 ，当且仅当他们剔除策略倾向的影响后在统计意义下 $com(A_1, A_2) < 1$，记作 $A_1 \prec A_2$ 。*

注意到这里体现了两个根据实践总结出来的初步结论：第一，算法具有不同的策略倾向性；第二，特定的策略倾向性之间存在特定的克制关系。同时我们根据常识判断该克制关系形成的不是一张DAG(Directed Acyclic Graph,有向无环图)。这一点可以通过用遗传算法对一特定算法持续训练得到的结果体现出来（只是我们并没有及时记录该方面的数据）。该现象形成的机理与机器学习领域的概念过拟合(over-fitting)似有相似之处。但是我们受限于能力，不能对该问题进行深入研究。

猜想 5.1 *搜索算法等效的传递性 若* $A_1 \approx A_2, A_2 \approx A_3$ *，则* $A_1 \approx A_3$ *. 若* $A_1 \prec A_2, A_2 \prec A_3$ *，则* $A_1 \prec A_3$*.*

可以发现我们能够在该猜想的基础上建立搜索算法间的偏序关系。

定义 5.3 *d 层暴力对抗搜索 我们将一个执行 d 层并在最后一层使用算法 V 计算权值的暴力对抗搜索算法称作* $BF_V(d)$ *。*

在这里，我们认为只考虑自己这一步所有情况为一层，考虑自己这一步与对方下一步的所有情况为两层。我们一般认为做参考用的对抗搜索使用的算法 V 为随机赋权值。记为棋盘未落子的位置随机赋权值的算法为 $RAND$ 。易知随机落子为 $BF_{RAND}(1)$ 。

猜想 5.2 *暴力搜索的偏序关系* $\forall 1 \leq d_1 < d_2, BF_V(d_1) \prec BF_V(d_2)$

这个猜想似乎是不言而明的。但是你并不能说明多搜索一层是否会带来一些误导。

定义 5.4 *搜索有效层数 若* $A \approx BF_{RAND}(d)$*，我们称 A 的搜索搜索有效层数* $act(A) = d$ *.*

5.2 搜索有效层数的进一步讨论

实际衡量的时候不一定需要完全不剪枝的算法即可得到近似的结果。比如可以引入 Alpha-Beta 减枝。

利用该概念实际预测两个AI的对战结果时，应综合考量其搜索有效层数和搜索倾向。

我们认为该定义的合理性在于他在尝试去描述一个算法的搜索空间在全部搜索空间中与胜率较高的方案的重合程度。在其中搜索方式影响了搜索空间的深度和广度，估价函数引导搜索方式决定出来的搜索空间和胜率较高的结构尽可能贴合，而估价函数的自动进化则是在估价函数的引导方向，并在一定程度上更改AI的策略。而这也解释了为什么我们经过遗传算法优化过后的AI出现了“特异性强化”，但是在面对其他AI的时候却效果没有那么显著。

5.3 策略倾向与战略布局

策略倾向 我们提出这个概念，是基于遗传算法的进化结果容易表现出极端在意防守或者进攻。而我们将其视为一种稳定的性质，是因为在面对类似的子结构的时候，该算法通常会有类似的决策。而进一步对对抗搜索进行分析，我们认为该倾向主要是由对抗搜索时预估函数带来的。而特定的策略倾向似乎会在搜索有效层数之外对胜负带来额外的影响。参考搜索有效层数的意义，我们认为这个倾向在搜索空间上表现为在多个有价值的策略分支上进行抉择时的倾向。

定义 5.5 *广义搜索算法等效 我们称两个该问题下的搜索算法 A_1 与 A_2 等效，当且仅当他们剔除策略倾向的影响后在统计意义下 $com(A_1, A_2) = 1$，记作 $A_1 \approx A_2$ 。类似的，我们称 A_1 弱于 A_2 ，当且仅当他们剔除策略倾向的影响后在统计意义下 $com(A_1, A_2) < 1$，记作 $A_1 \prec A_2$ 。*

我们认为，特定的策略倾向性之间存在特定的克制关系。同时我们根据常识判断该克制关系形成的不是一张DAG(Directed Acyclic Graph,有向无环图)。根据前文对策略倾向的描述，我们认为存在一种可能的量化评估手段：总结出一些特定局面，并根据算法在该局面下的决策的攻击性进行打分，攻击性的评估依赖于更为有效的智能算法。该现象形成的机理与机器学习领域的概念过拟合(over-fitting)似有相似之处。但是我们受限于能力，不能对该问题进行深入研究。根据该定义，还可以类似地提出广义搜索有效深度，这里就不赘述了。

战略布局 与围棋和象棋不同，战略布局在五子棋入门乃至初级阶段重要性都不是特别大。但是我们发现在开局阶段这样的思路还是很重要的，而我们认为这是难以通过增加搜索有效层数来进行优化的。抽象出来，战略布局这一概念对于一个问题是否有意义，关键在于该问题是否可以被分为不同的阶段，并且不同的阶段目标略有不同。具体到算法设计上，我们应该通过对不同阶段设计不同算法，在同一阶段内讨论搜索有效层数；阶段的划分也许更应依赖人类在该问题上的经验或者对算法进行细致分析。根据搜索有效层数，我们认为对抗搜索其实还是在暴力且贪心地去寻找局部最优解，因此在应用对抗搜索的算法中也应以此为划分阶段的依据，让对抗搜索成为整个流程最基本的单元。

5.4 理论的适用范围

普通的搜索有效层数只能在不关注策略倾向与战略布局的算法上适用，比如前文描述的 $BF_{RAND}(d)$ 。而广义搜索层数可以在不关注战略布局的算法上适用，但是其量化评估仍然是个问题。

具体来说，一系列算法是否适用，可以通过测试搜索等效的传递性是否满足来判断。因为这只是一个近似的模型，所以数值上大致满足即可。

我们发现这次研究出来的成果完全可以用这几个概念进行阐释。而对这几个概念的深入思考会启发我们有关这类算法设计思路的不足：仅仅是立足当下而缺少全局布置。这暗示我们在设计类似决策算法的时候，应该事先利用人类智慧设置若干阶段并为不同阶段设置略有不同的算法，或许能在实际问题中获得更好的结果。

6 总结

6.1 研究成果

1. 完成了一个轻量级的具可拓展性的五子棋对战测试平台的开发；
2. 实现了若干算法并对算法的实际效果进行了实验测试；
3. 提出了搜索有效层数作为对研究涉及的算法的总结并探讨了该概念的前提、必然结论，同时利用该概念对实现的算法进行了更深入的解释。

6.2 利用对抗搜索进行算法设计

对抗搜索的本质还是较为贪心与暴力的。因此进行设计之前需要先对问题进行阶段划分，使得每一个阶段都足够小到可以认为局部最优解在这一步可行，使得具体算法成为整个流程上的小单元。一个能部分体现此原则的例子是依旧使用一棵搜索树，但是在该搜索树的不同部分设计不同的估价函数。而事实上已经有关于此思路的研究了。

具体到流程的一个阶段，可以发现估价函数、遗传迭代、对抗搜索各有其独特的优势。可以利用遗传迭代研究优化估价函数，并利用合理的估价函数结合对抗搜索可以实现较好的效果，其中的关键是如何高效地计算估价函数并利用估价函数给出一个较为合理的搜索空间，利用搜索来实现战略上的布局和局部的优化，而单纯去计算估价函数或者单纯去搜索都会导致不能达成较好的结果。这些算法都较为轻量级，便于理解易于调试且有较好的拓展性。

同时，针对对抗搜索，还可以使用搜索有效层数进行效果量化评估。

6.3 进一步研究展望

1. 在理论中还有大量的猜想和假设。我们希望下一步进行试验获取数据来说明这些问题以巩固理论的事实依据。
2. 针对五子棋问题人类已有知识进行深入研究，并以此为依托设计一个强大的算法。
3. 将提出的利用对抗搜索进行算法设计的思路运用到实际问题上。

4. 进一步开发五子棋对战平台，增强扩展性与可用性，优化代码，为他人算法学习提供一个实践平台。

5. 我们这次仅研究了部分初级的算法，对于更现代的模型优化手段例如神经网络，以及更先进的棋类搜索手段例如DF-PN search，限于时间、能力，都没有涉及。我们希望将来能对这些方面进行研究。

7 鸣谢

本课题的顺利开展离不开母校和老师同学的支持。感谢母校江苏省常州高级中学、班主任危安琪老师对我们的支持。我们更要特别感谢指导教师曹文老师在这个过程中提供的指导及活动相关支持。

8 附件

1. 陽齟齾v0.7.4.zip[5]

2. 程序框架说明文档.pdf

3. 对战平台使用指南.pdf

参考文献

[1] 网络资源http://ku10.com/#415

[2] Victor Allis,Searching for Solutions in Games and Artificial Intelligence[D],Holland:Maastricht University,1994[6]

[3] János Wágner István Virág,SOLVING RENJU[J],ICGA Journal,March 2001

[4] 任之洲,浅谈启发式思想在信息学竞赛中的应用[A],余林韵,2015年信息学奥林匹克中国国家队候选队员论文集[C],中国计算机学会,2015[7]

[5] 罗文浩,五子棋对弈平台的设计与实现[D],西安:西安电子科技大学,2015

[6] 张效见,五子棋计算机博弈系统的研究与设计[D],合肥:安徽大学,2017

[5]可至gitee或github获取，下同。
[6]我们在马大官网找到了该论文
[7]该论文可能不易获取，如有需要可联系我们