

人工智能报告

**姓名：陈悦莹**

**学号：51203096994**

**组名：A+**

**日期：2015/1/17**

目录

[1.围棋 3](#_Toc409275496)

[1.1起步 3](#_Toc409275497)

[1.2思考 4](#_Toc409275498)

[1.2.1 对于uct的思考 4](#_Toc409275499)

[1.2.2 用排序筛选代替剪枝 5](#_Toc409275500)

[1.2.3为什么我们放弃了amaf 6](#_Toc409275501)

[1.2.4 pattern和trick怎么组合才有效 6](#_Toc409275502)

[1.2.5 那些除去策略之外可以提高棋力的方法 6](#_Toc409275503)

[1.2.6 怎么判断棋力 7](#_Toc409275504)

[1.2.7 针对比赛该做些什么 7](#_Toc409275505)

[1.3 实现 8](#_Toc409275506)

[1.3.1 放弃自己预想的数据结构 8](#_Toc409275507)

[1.3.2 棋盘的可维护性有限 8](#_Toc409275508)

[1.3.3真气和伪气的取舍 8](#_Toc409275509)

[1.3.4在减少时间开支上那些看似有效的方法 8](#_Toc409275510)

[1.3.5 最终没能实现的gpu编程 9](#_Toc409275511)

[1.3.6 突破瓶颈 > 找到正确的瓶颈 > debug > 加速 > 实现 9](#_Toc409275512)

[1.4 反思 10](#_Toc409275513)

[1.4.1 获得 10](#_Toc409275514)

[1.4.2 遗憾 10](#_Toc409275515)

[2.人工智能 10](#_Toc409275516)

[3.总结 11](#_Toc409275517)

# 1.围棋

## 1.1起步

全组四个女生，全部都是没有任何围棋经验的。这里的没有经验是指

1.我们无法判断一步棋的好坏与质量

2.不能通过人机对弈的方法来找出不足

3.因为不具备其余前导知识我们从思路上从一开始就很依赖模拟—搜索这个模式。

不管是我们的ai还是我们自身，从一开始我们就在想着同样的事情。我们的起步很低，因为一点围棋知识都没有的我们要怎么教一个同样什么都不懂的电脑呢？这种起步的好处在于我们并没有因为自视棋艺高而批判某种算法，坏处在于从一开始我们就只能思维定势在这一方向上。

至于我们为什么没有事先去学习围棋，我个人的情况是：在本学期有打算学围棋，虽然最后的成果只有知道了围棋规则，这种水平自然不敢妄言会围棋了。直到这门课结束，我依旧没有下赢过围棋小游戏。

还有一件事，就是关于coding能力的问题，关于这一点，开始的时候也挺自以为是的，觉得自己的起步还算不错，但是也确实因为这份自负给团队带来了一些麻烦，之后我会详细说明。

最后一点是思维上面的起步，这一点很重要，这一点决定了我们之后AI前进的方向。

从我的角度来思考，在我只知道规则的情况下，我要怎么去下围棋呢？

1.观察有经验的对手的下棋方法

2.“下一步猜三步”

至少我觉得这是一个下棋的通式，如何猜中对手的心思和如何比对手想的更远。

至少从这种思路切入的话，很自然的想到两个方向1.模仿学习 2.搜索解空间。前者应该需要用到机器学习的相关知识，但是这个方向我一开始就没打算继续，首先是因为自己这领域的知识太少缺少入手点，再者也是考虑到难度和课程时间，怕技术上无法实现得不偿失。后者应该是计算机拿手的，人在有限时间内能搜索的广度一定比不上计算机，至少从数量上来说是的，质量上就不好评定了。

至此就确定了方向，接下来我们的目标就是两个

1.提高计算机的搜索广度

2.提高搜索质量

## 1.2思考

### 1.2.1 对于uct的思考

模拟-搜索模式是顺势而生的一个想法，我想也是大部分人都会选择的方法。果然能找到更为完整的前人的工作——uct。

其实我对于uct的认识或者感情随着我自身棋力的变化差别还挺大的。从一开始的不得不依赖到质疑再到最后觉得神奇。因为分工原因，uct的基础源码并不是我实现的，uct也不是我从paper上学来的，而是我之后接手项目继续开发debug的时候直接从代码上读出来的。

不得不依赖

当我的围棋水平和一个只知道规则的电脑一样的时候，我不得不依赖uct，因为就算是傻瓜式的搜索计算机下的uct一定比我考虑的情况多得多。一开始自己是弱于uct的，在大量对站里uct下在哪里，好不好，我都没办法判断，就比如有一个明显的点下在那里可以吃掉一大片子，但是uct没有下，我也只能认为uct比我有更远的考虑。（其实这一时期因为气的计算错误，那段算错气的阶段里……我一度说服自己uct看起来就是在随机下棋一定是uct有更深的思考……）。后来用纯uct（气已经没有问题了，是正确的uct），不加任何质量上的筛选单纯傻瓜式的uct，我观察了很多局之后，自己摸了些门道之后，也有“uct和我想到一起去了”和“比起uct我更会选择下在这里”的情况。所以前期几乎就是uct教我下棋，再加上当时经过提速的纯uct已经打败了上一届第三，所以一开始还是挺依赖的。

质疑

质疑应该是一开始就有的，只是碍于自己水平的问题，就算有顾虑也没有更好的解决方法。如果围棋问题仅仅靠着搜索就能解决那就不叫难题了。围棋的规则太过简单，以至于太活，很难有有效的方法去缩小他的搜索空间，所以搜索空间的巨大是其他棋不能比的，我们单纯uct中场搜索也就只能到达4层左右，毕竟指数级增长。

后来我的棋力增长超过uct的时候，已经能很明显看出uct下的一些臭棋了，这时候这种质疑就更深了。后来我脱离代码去看了之前同组同学找到的两篇相关论文，两篇论文的差距就在于访问次数和得分两者所占比例的不同。这个比例的控制我一直觉得很神奇，当然论文里面的这个比例是有大量实验支持的，我们一共换过三个uct公式，其中一个提高了棋力，另一个没有。论文多以19\*19的棋盘做研究基础，而我们是13\*13。我觉得这也很可能是论文公式所述效果并没有在我们实践中发挥所述那么好的效果的原因。后来一度想要调整，但是当我发现这才是我们的瓶颈点的时候已经接近后期了，在没有任何理论支持下，我没敢轻易尝试。最后选的是三个公式中最好的那一个。

还有uct的收敛问题，这个是组里另一个同学在后期发现的，发现我们在搜索到9.5s 20k盘左右的时候就已经很容易收敛了（当时我们的uct是9.5s 120k），当然需要排除一下一开始子太少情况下难收敛的情况和有明显好点出现时很快收敛的情况。这对于我来说是个很震惊的消息，因为我一开始就定式思维在：搜索愈多愈好。

一开始没能提出来的顾虑后来一个接一个的变成瓶颈出现了。就我个人而言，我觉得这是我们这次项目的致命伤，我们对uct的质疑没有更早的去验证，直到最后才发现这里才是最大的坑，最大的瓶颈。

神奇的uct

最后我觉得uct的神奇在于，在我发现了这么多uct的局限性以及模拟那么多盘都是浪费之后，试图在它上面做改进（包括剪枝，pattern，一系列trick配合筛选），企图把傻瓜是下棋变的聪明一点，提高搜索质量，但是结果发现效果竟然都不如纯傻瓜式uct。

总之uct带给我太多惊喜，一言难尽，最后比赛时我们还是用的纯uct，表现也就不上不下。感觉又回到了最初，不得不依赖uct。就算比赛结束了，我依然觉得我们的棋力无法提高的瓶颈在uct，但是我没有找到解决他的方法。

### 1.2.2 用排序筛选代替剪枝

剪枝一般用来缩小搜索空间，能更快的收敛到最优解，围棋搜索空间巨大，感觉采用剪枝也是理所当然的。要减去的枝其实也很明显，简单来说

1. 开局明显空旷的地方
2. 已经无力挽救的地方

不考虑复杂情况就是因为围棋太活了，如果把剪枝条件限制的再严一点，很容易就陷入局部最优解，对全局而言未必好。关于这一点，我能断言是因为我确实测试过。

仅仅是减去上述这两部分对搜索空间来说都是很大程度的减小。而且这两个条件明显所涵盖的点多并且条件松，所以我们都预计这个带来的负面影响将小于他正面的提高。之后我们确实付诸于行动了。但是从实际测试情况，和我们的预期是相反的。原因出在

1. 每一步uct搜索都需要重新设定棋盘状态并分析
2. 每一步剪枝所用的trick都需要重头做一遍
3. Trick中大多使用真气
4. 围棋的前中后棋剪枝条件并不相同，比如中后期放弃明显空旷的地方其实效果并不好

在有限时间里，每一步都浪费一点时间，累加起来是不可承受的。我们想的折中方法是在uct启动之前传进去一个我们认为好的搜索顺序，让uct在这个顺序上启动搜索。要点是：

1. 这个数组针对每一次ucts搜索，即只在每次uct启动前更新这个价值点排序的数组，只在真实发生的棋子移动后这个数组的顺序和大小才会改变。
2. Uct里面模拟的时候都按这个数组中优先级来搜索
3. Uct模拟的时候可以记录dirty位，即已经在先前模拟中被占据已经属于不合法的移动，不会产生模拟在不合法位置的情况

在这个方法里我们没有减掉任何一个点，并且一切价值点的判断放在uct之外，那这里的时间开销就不足为惧了，并且给了我们更多施展的空间，我们怕局部最后而减去不该减去的点，但是我们不用担心提前用uct模拟局部最优的点。而这也达到了我的目标，尽快收敛。

我们对以下情况的点给予较高的优先级：

1. 吃子点
2. 救子点
3. 提气点
4. 棋谱匹配点
5. 距离对方上次下子点近的区域

我们对以下点给予较低的优先级：

1. 棋盘最边缘一圈

实验结果，这个措施提高了我们的棋力。在我写报告的时候觉得这一部分还有很大优化空间，对于优先级的分配应该可以按照围棋进行时期不同而不同，每一步选择可以更为细致一点，比如救子点，我们做到的是能保证救子不是白救，就是救了这个子之后这个棋串的气变多了。

### 1.2.3为什么我们放弃了amaf

我认为amaf和uct有点相悖，amaf重视全部模拟下这一点的分值总和，uct则重视每一步这一点的分值。根据上一届同学(undecidable)的经验，这两种方法虽然相悖，但是如果可以一起使用，在不同时期控制两种方法贡献的比例可以带来很棒的效果。但是我们最后没有用进去，是出于很无奈的原因，很大部分是代码实现上的问题，amaf要求维护一个全局数组，但是我们当时已经用了多线程加速uct，多线程维护一个数组必然会遇到线程加解锁问题，强行把amaf放到多线程当中，只怕频繁的分值更新操作会让多线程变成单线程。我们最后选择了保留多线程而放弃了amaf。

### 1.2.4 pattern和trick怎么组合才有效

关于pattern有很多论文研究，我们也在论文中找到了一种配合方式。这个在总报告中有详细叙述，而且也不是我做的，我这里就不赘述了。单用Pattern的效果其实并不好，后来尝试配合，但是我们的uct并没有优化加速到足以支持这些时间开销，和剪枝的效果差不多，这些组合带来的效益并没有弥补uct搜索盘数减小带来了负面影响。所以pattern和trick怎么组合才有效，我觉得论文还是最好的材料，如果uct的优化足够倒是不妨一试。

### 1.2.5 那些除去策略之外可以提高棋力的方法

1. 开局定式
2. 棋谱
3. 置换表
4. 位运算加速

这些方法有好有坏，之后我会具体讲，有些方法只是想得好。

### 1.2.6 怎么判断棋力

对战是比较直接的方法，比如模拟对战10局，达到80%以上胜率就可以说胜者棋力较高，不然胜负可能只是一定概率而已。重点是怎么找这个对手，我们的经验是：

1. 格外trick少，比如提前吃子救子这种trick
2. 棋力稳定，应该是遇强则强
3. 具体需求

胜负终究是双方棋力博弈的结果，trick可能会带来一些效果，测试的时候容易被trick混淆掉真实棋力，而且因为trick容易针对，就算赢了这一版，很可能是因为对他的trick做了针对处理，而并不是本质棋力的提高。

我们测试前期是以上一届来作为对手，从最后慢慢开始往上走，最后阶段我们的对手就是上一届第二名。

我们也有自己不同时期版本之间比的时候，但是我觉得真的有效的是在uct换公式那次。其他时候单变量变化带来的效益不明显的时候对战的意义都不大。

### 1.2.7 针对比赛该做些什么

我觉得比赛比的不仅是棋力还有我们的头脑策略。当棋力旗鼓相当的时候，比的就是trick和运气了。也是在正式比赛之前和别人模拟比赛之后得出的结论：

1. 要靠吃子来抢占先机。这个其实特别明显，在前20步的时候，大家的uct起的作用并不那么抢，但是如果在这个时期能吃掉对方的很多子，就可以造出很多假眼，这些假眼很有潜力成为真言。而且还有更重要的原因，因为我们是13\*13的小棋盘，如果被抢先吃去5+的棋子，先机就会被抢走，棋面一下就会弱下去，靠后期是很难力挽狂澜的。
2. Trick最好不要全程使用，trick是用在uct搜索之前的，如果全程使用，就会因为trick的优先级对uct的效果大打折扣，这种折扣不止是体现在当下。
3. 最后的收官。这个是用在最后的时期，胜负几乎已经确定的情况下，要尽可能占据大的地盘，这对最后难分胜负的对弈很有效。这个最后遗憾我们没能加上去。

## 1.3 实现

### 1.3.1 放弃自己预想的数据结构

这点是我对全组感到抱歉的地方，我负责一开始基础工程的建立。我习惯性的按自己的编程习惯封装了一些对象。后来经同组同学测试，单单这个数据结构就使得棋盘效率降了一倍。我自己也很吃惊，然后反复看自己写的东西，不说效率降低的问题，再回顾就觉得自己写的欠考虑，没有想的很清楚就动手写了，结构有冗余的地方并且耦合性很高，无论是后期增删还是和别人合作都是不利的。后来全部推掉重写了。用了最简单的数据结构，就只封装了棋盘状态一个对象。这期间浪费了一个礼拜的宝贵时间，挺耽误后面同学的开发的。我的经验教训就是：动手之前想清楚，除非已经对这个工程的脉络清楚到不行，不然就使用简单的数据结构，对自己对团队都好。

### 1.3.2 棋盘的可维护性有限

棋盘上的维护泛指对棋局的维护，比如棋串维护等。因为分工关系我有部分代码实现工作在这方面。但是我发现这边能做的并不是很多，比如很明显有效的并查集维护棋串，合并循环，减少循环等。这个维护和数据结构有关系，因为我们的数据结构很简单，又加上用了空间换时间的方法，就更一目了然。还有位运算这种，位运算确实快，但是他快的运算我们多半没用上。换的性价比不高最后舍弃了。

### 1.3.3真气和伪气的取舍

气是围棋很重要的一部分，很多操作都要求真气，我们组还有很长一段时间因为气算错而陷入泥沼。我们最后保留了真气和伪气两个，真气耗时，伪气只能判断是不是只有一口气了。就像我之前说的，任何一个在uct里面使用的耗时操作我们都一再斟酌，再少的时间消耗都禁不起大量的堆积。配合我们最终的策略，我们在uct搜索之外使用真气，在uct之中只用伪气。

### 1.3.4在减少时间开支上那些看似有效的方法

1. 置换表

这部分不是我实现的，全靠我们组另一位编程大神（刘静静）实现。实现过程很辛苦。当时从数据上置换表能达到1/10中率，其实是很能缩小重复局面搜索的。但是万万没想到构建置换表使用他维护新的数据结构所花费的时间更多。得不偿失的又一个例子。我知道有别的组用了这个方法，并且可能对提高棋力有效，但是对于我们当时的策略下，置换表得不偿失。

1. unrolling

这个是我为了加速c语言操作查找的，大致是利用流水线操作知识把循环拆成两个来加速。但是实际上这种东西很过时，因为vs在编译的时候会帮助优化，而且优化的远比这些操作来得多，很明显的一个例子就是release版本和debug版本之间的10倍效率差距。

### 1.3.5 最终没能实现的gpu编程

因为我一直觉得在uct里面各种pattern和trick的组合优化不好是因为我们还是不够快，不足以支持这些操作，以至于无法让这些东西发挥效果，所以一直在想办法怎么还能再快一点。这个算是我最后的挣扎了，在临比赛前一天考完试之后开始做的。有这个想法倒是之前就有，因为听说GPU计算比CPU计算在某种情况下快300倍。这个对我诱惑力很大，然后开始尝试做，因为GPU编程之前没有接触过，所以光是入门就浪费了不少时间，但是真正没做下去是因为我发现有些问题：

1. 对于我们单个节点的产生并不花费多少时间，并且计算量只有计算分数
2. 我所认识的GPU编程是：原本在CPU上面做的运算，在GPU上开辟同样的空间，复制数据，做完之后再把结果复制会CPU。所以，究竟计算缩短的时间有比复制数据的时间长么？如果复制数据的时间更长，那GPU完全是个累赘。
3. GPU编程的.cu文件和c文件的兼容性差，虽然是有改变编译命令行的方法可以解决，但是特别麻烦。

不过说到底，我最后并没有把这个尝试真的变成现实，也是因为自己编程水平有限，只剩一个晚上实在难以做出来，而且之后遇到的问题都在我更不熟悉的领域了。所以最后放弃了。但是还是存在一些好奇，GPU编程能很好的加速么？

### 1.3.6 突破瓶颈 > 找到正确的瓶颈 > debug > 加速 > 实现

这是这次写程序最直观的难度感受。

实现

实现角度而言，因为有Brown作为基础，都不需要考虑交互的问题，各种trick pattern uct棋谱等又是独立单元，每个单元的也没有什么实现难度。

加速

普通加速其实很简单，都不能算作瓶颈，应该是大家都能想到的解决方法。只要稍微分析一下耗时的地方加以优化就行了。

Debug

Debug一直都比写代码难，但是这次格外觉得难的原因在于小单元组合太多，小问题太难发现了，就比如我们组气计算错误，真的是很小很小的一个问题，但是却困扰了我们很久很久。而且本身调试也不是很简单。一些逻辑上的bug只有通过对战发现异常，还需要人为分析。我自己是直到最后都没找到一个觉得诚心的调试方法。

找到真正的瓶颈

我觉得这是我们这次AI的致命伤。我们一直以为搜索数量是我们的瓶颈，但是直到最后才发现UCT公式本身就还是个瓶颈。这次我们做了很多的尝试去加速，用了很多的方法，有成功有失败，但是如果我们能早些发现瓶颈所在，应该还能再努力一下。通过这次大作业，我感觉找瓶颈这件事应该是：不要完全信任任何一个决策。我觉得要分工每个小细节都去做，其实也是很难的，工作量会剧增，说不定每个方向上的突破效果还会打折。更快的找到瓶颈可能需要更多对围棋AI的理解吧，我们太过依赖搜索-模拟这个模式了。

## 1.4 反思

### 1.4.1 获得

这次大作业真的很有趣，无论是从思考上还是实现上对我都是一个挑战。但是在这次大作业中收获的确实很多。在整个过程中，自己能明显发现自己是一个手快于脑的人，因为这种原因也给团队带来不好的影响，并且做了大量的无用功。在真正开始做AI之前我重视实现高于制定策略方法，但是真的做起来就会越发感觉，实现只是个基本，思想才是核心。我最后整理比赛时用代码时，删掉了大量最终没有使用的代码，这些代码都花了心思，甚至比留下来的代码要复杂很多，但是都没用。剩下来的代码显得那么简单，很难想象我们是怎么把一个简单的Brown加工到那么臃肿的，感觉绕了很远的一个圈，如果之前能再想的清楚一点的话，我们应该可以事半功倍。在coding能力上，我也获得了很多，感觉c语言确实很难处理，在过程中也有因为自己实现不了而放弃的尝试。然后是团队合作，真的很开心四个女生能组队，我们四个本身就熟识，而且真的都是非常靠谱并且行动力很高的人，真的是一次发挥各自所长的合作。这是我至今为止最愉快的一次大作业合作了。

### 1.4.2 遗憾

这次大作业我一直感觉不上手，就是感觉做了很多但是效果达不到预期，总觉得做偏了，但是又找不到正确的方向，做的好不好和做得多不多并没有什么关系，关键在于能不能作对点。所以我一直感觉虽然在做，但是一直都做在不痛不痒的地方。遗憾总是有的，就是没能快点发现问题所在，让大家的努力都没能发挥它应有的价值。还有就是一些想法没能实现出来。最后比赛成绩占中游，我们都觉得是能接受的一个结果。

# 2.人工智能

人工智能这门课一开始就让我们写了对人工智能的看法。我当时觉得人工智能是建立在理性思考和决策之上的。上完这门课之后这种感觉更深了，我觉得人工智能中所包含的方法都有很强的数学性。而数学又是被太多次证明存在在我们生活的方法面面，存在在万物构造之中。恰好在这门课结束时，在知乎上看到一篇文章是说在围棋AI中应用神经网络等能够打败6-8段围棋选手。我们的大脑比我们聪明，而目前我们比我们的电脑聪明。所以我觉得之后会有人工智能和我们之间互相的竞争吧。

# 3.总结

人工智能这门课是我目前上过的最有趣的一门课，后期的人工智能因为数学性太强我觉得挺难的，最后的大作业很有趣但是也很累，但是也获得了相应的成就感。真的没想到自己最后的报告能写那么多，大概是每个坑对我来说都是新体验吧，印象太多深刻了。最后感谢一学期悉心教导的张教授，还有悉心解答的助教，还有就是一群很靠谱的队友啦~