报告大纲：

1. 前言（球）
2. 技术路线
2.1. Uct（球）
2.1.1. 剪枝（球）
2.1.2. 模拟策略优化（pattern）（二慧）
2.1.3. Amaf（二慧）
2.1.4. 置换表（静静）
2.2. 棋谱定式（静静）
2.3. 其他优化方法
2.3.1. 位运算（静静）
2.3.2. 并查集、气（球）
3. 过程
（陈悦莹）
4. 总结

模拟策略优化

通过添加领域知识，相比于原来的传统随机模拟来说，好处在于模拟看起来更具战略。在[1]中的 pattern 和其他 pattern 不同的是，pattern 仅被用于在找局部解当中，并不一定是全距最优解。因为在蒙特卡洛模拟中，找到一个更优的序列比找到更优的一步棋更为重要，如果将 pattern 应用于全部棋盘，反而会降低准确率。

我们仅仅考虑再最后一步棋附近匹配 pattern。因为局部最优解最有可能成为最后一步解。这样最后的局部序列就得到了。

首先辨认是否是 atari。如果是的话，就可以 saveing move 否则在最后的一步棋周围 8 个位置寻找 capturing stone，最后，随机下棋。

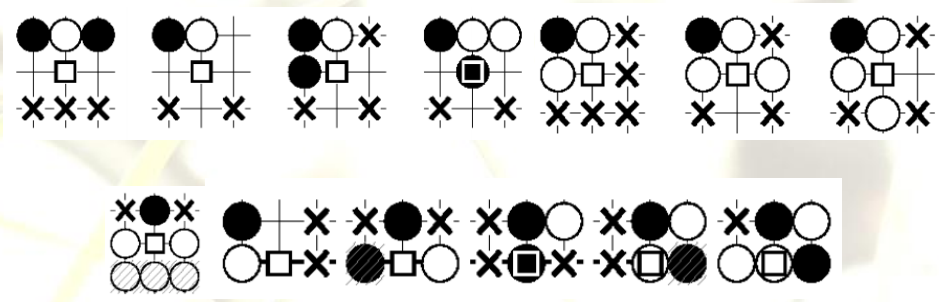

图 1. Pattern 图示[左上四个为 hane，右下四个为 border，其余为 cut][2]

Patern 部分的实现难点主要在于坐标转换与判别，在我们的设计中，根据最后一步子周围八个点是否与设定的 pattern 匹配来进行，匹配过程中不考虑转换问题，坐标转换部分。

除去 pattern 之外，其他的策略也十分重要，在[1]中，作者综述了 6 种 pattern，分别是：

1. Atari defense move：救子，遍历棋串，找到只有一个气的棋串，决定救不救。
2. Nakade move：防止对方做眼，如果有三个连续的空位置，并且被对方棋串围着，则下中间的一步
3. Fill board move：找到棋盘上的空位置下
4. Pattern move：与 hane，cut，border 等模式进行匹配并下棋
5. Capture move：如果对方棋串仅剩一口气则吃掉
6. Random Move：随机产生一步，如果是合法位置则下在这一位置

而在实践中，我们发现如果棋走的越分散，棋力会有明显下降，所以就去掉了 fill board move，而且由于加入 pattern 过于影响速度，最后也并没有全程开策略。

UCT-RAVE 方法

UCT-RAVE 是 UCT 和 RAVE 两种方法的融合，就 UCT 来说，需要根据 MC 模拟，推算当前状态下最佳的步骤，也就是赢率最大的一步棋，这样的话，对每个节点都要模拟成百上千盘棋局，在有限时间和计算能力下，得到的信息是非常有限的，而 RAVE 则关注于相关节点之间的胜率关系，从而得到一些相关信息，尽管这些相关信息可能并不十分准确，但能够得到一个较准确的估计值。

RAVE 是根据 AMAF 启发式权重进行搜索，这种方法的本质是在于，对于每步棋都有一个权值，这个权值不随执行这步棋的时间改动。其实说这种方法是违背围棋精神的，毕竟在实际的对战中，每一步都是具有很强的时间敏感性的，比如说现在救子吃子都很重要，然后选择了吃子，如果把这一步救子的权值累加到下一步，就会有一个问题，因为再过一步，救子没救成，已经被对手吃掉了，然后之前的权值其实已经一点价值都没有了。这样在战场上无疑是很致命的错误。

为了扬长避短，UCT-RAVE 就是汲取这两种方法的优点，通过对短期权重和长期权重的平衡，得到的一个权值。RAVE 方法的学习能力要比 UCT 快，但是准确率没有那么高，UCT-RAVE 呢，则是通过将快速学习得到的全盘权重，和 MC 模拟得到的准确的实时权重结合，从而得到一个较理想的权值。

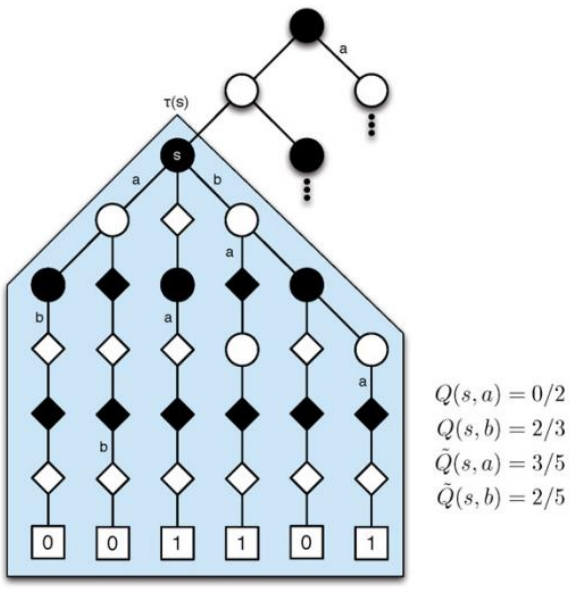

图 1. RAVE 图示[2]

具体的 UCT 和 RAVE 比较可以参见图 1，其中 s 是当前棋盘，a 是当前设置权重的位置，$Q(s,a)$表示 UCT 胜率，$\widetilde{Q}(s,a)$表示 RAVE 胜率，$Q_*(s,a)$就是加了权重的 uct-weight 和 rave-weight。

$$Q_*(s,a) = \big(1-\beta(s,a)\big)Q(s,a) + \beta(s,a)\widetilde{Q}(s,a)$$

其中$\beta(s,a)$的计算公式为：

$$\beta(s,a) = \sqrt{\frac{k}{3N(s)+k}}$$

其中 k 是设定的调整权值的参数，当模拟盘数为 k 是，UCT 胜率和 RAVE 胜率权重相同。

UCT-RAVE 的公式就可以表达为：

$$Q_*^+(s,a) = Q_*(s,a) + c\sqrt{\frac{logN(s)}{N(s,a)}}$$

其中 c 是经过调整的参数， 关于 c 参数的调整在 uct 部分已经介绍过，这里就不再赘述了。

真正在实现的时候

个人报告

个人贡献：

简单来说，我主要做的工作有：

1. UCT-RAVE 实现

   在 UCT 之余，我们还实现了最基本的 AMAF 方法。在基本调研之后，我们觉得 UCT-RAVE 方法应该可以提高围棋的战斗力，因此决定将其添加到现有的 UCT 中，我通过修改原有代码，得到了数组实现的 RAVE，但是发现这样和原有的节点结构就冲突了，而且在多线程中，数组结构会因为读写冲突报错，当时检测了一下单线程状态下的 UCT-RAVE 的表现，差强人意，所以最后在实际对战中就放弃了这一块。

2. 策略优化

   策略优化是在 uct 搜索中，为了增强模拟对战的真实性、有效性，双方对战所采取的策略，我通过查阅相关资料，根据文献中总结的策略优化经验，敲定了 hane、cut、border 等 pattern 和，atari denfense, fill board, nakade, capture 等策略。并实现了其中的一部分。

在真正模拟中的策略

3. uct 公式优化

uct 的公式参数调整对其的表现十分重要，也就是 exploitation 和 exploration 的比重问题，在[1]中综述了集中 UCB 公式，我们也是一个个试了一下，最后敲定了 UCB-Tune2 来作为最终的公式，在我的检测中，在 exploitation 方面的权重在开始时可能很大，但万步以后，就很小了。

4. 测试与调试

这部分工作虽然看起来不值一提，但确实是很重要的一部分，比如说在比赛前一天的时候，我们想出来了一些小 trick，排列组合得到了十几个版本，我们几个就来回测试，版本之间比，版本和上一届的程序比，和其他组的程序比，最终敲定棋力最强的一个版本参加比赛。

感想与感受：

人工智能这门课可以说是这个学期最有趣的课了，之前选这门课的时候就知道最后会有期末大作业，每个组做围棋 AI 最后还有对战！虽然也能想象出来为了优化最后会很辛苦，但还是坚定的选了这门课。

平时的理论课程看起来和大作业部分没有什么关系，

其实感觉大学学了这么多课，很多都是考前背背书就过了，虽然也有大作业，但一般也不是竞技性的，不会有个量化排名，大家自己做自己的，辛苦一段时间做出来，展示一下，最后一般给分也都不错。感觉这次大作业最大的特点就是，不是做得多就做的好，经常辛辛苦苦实现一些东西，最后用了这些优化方法的版本反而不如没有经过优化的，所谓事倍功半，也就是这种感觉了。

我们组算是比较提前做的了，之前 uct 做出来，就写好了吃子救子 pattern 等策略优化，在比赛前一周的时候跟关系比较好的两个组比了一下，能赢个 60 子+的，感觉很是不错，不过当时这两组的 AI 也都不是很完善，过了几天过后，其中的一组就过来草虐我们了。再有就是比赛前两三天的时候，跟"天元圣手"他们组比了一下，被屠城了，当时心情差的都不想复习了，所以后面也试图做了很多像和他们对抗一下，但最后也没有的到很好的成果，在最终的比赛中也确实输了，虽然很伤心，但也算意料之中。

其实做优化比较难的一点也就是不知道怎么去衡量，优化了的是不是就是好的，比如在 11 版本上，我们做了 9 个版本，做出各种调试，其中大概四五个版本都是超过前一个版本的，但在对战其他组的时候，表现相差无益，虽然有时能赢天元圣手，或者上届的 undecidable，但是也都不稳定。

另外感觉通过这种竞技性大作业，也不仅仅是闭门造车那么简单，比如说，需要时时关注别的组的进展，并通过对战来相互提高。比如说跟"神之一手"，在我们 version 的时候，跟他们比，可以赢 60 子+，过了几天，他们跑过来要求跟我们一战的时候，

就可以赢我们 30 子+了，当时心里虽然很不爽，但是这种受挫也给了我们继续优化的动力，等到我们 version 的时候，就可以先后手稳赢他们了。

作为一个纯妹子组，实话说相比汉子组还是有点劣势的，比如说熬不起夜，但是相信我们的优点也有很多，比如说认真仔细，准备充分。

最后，还要感谢张老师的谆谆教诲，助教的热心指导，和同学们的关心与帮助，如果没有这些师长，同学的帮助，我们肯定无法取得现在的成绩，谢谢你们！