明棋规则军棋 AI 项目报告

范舟

2016 级 ACM 班

2016 年 11 月 27 日

目录

1 设计思路

1.1 AI 模式的选择

军棋是一类规则较为复杂的棋类游戏，在设计 AI 算法时，如果采用设计专家系统的模式，需要人工分析棋局中的各类不同局面，逐一设置决策，不仅十分繁琐，且对设计者的军棋水平要求较高。因此，笔者认为，对于棋类游戏较为通用的搜索算法是更易实现的方式。

此外，笔者此前曾经开发过一个简单的五子棋 AI，对于棋类游戏的搜索算法有一些浅显的了解，但与五子棋相比，军棋的规则更加复杂，对于搜索算法的设计有更高的要求。

1.2 搜索算法的基本框架

AI 结合估价函数，采用“最大-最小”（Minimax）原理在以当前需解决局面为根节点的决策树上进行深度优先搜索。

- 估价函数：为棋局的某个局面计算出相应的估计评分，这个评分表示此局面对己方的有利程度，数值越大表示越有利。
- “最大-最小”（Minimax）原理：设置一个深度优先搜索的终止深度 K，从根节点（当前的局面）出发，在搜索的每个节点上枚举可能的着法，将每个着法产生的新局面作为子节点，进行下一层搜索。到达终止深度 K 时，调用估价函数对局面估值并返回。若规定根节点为第 1 层节点，则奇数层节点产生的是己方的着法，在子节点中选取得分最高的一个；偶数层节点产生的是对手的着法，在子节点中选取得分最低的一个。

在搜索中，根节点得到的最佳着法便是 AI 对于此局面作出的决策。

2 细节实现

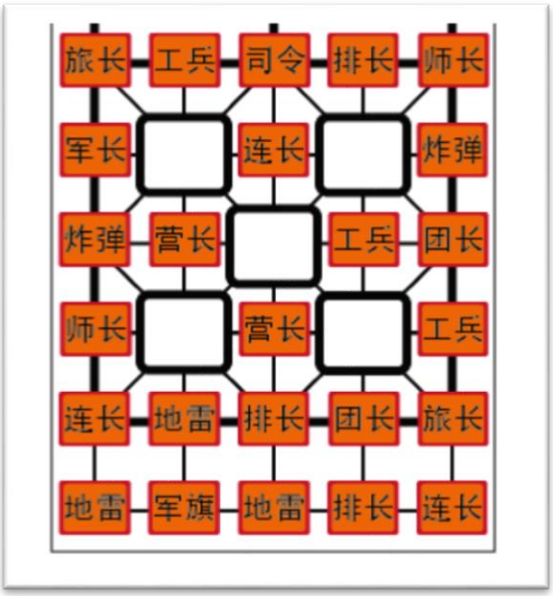
2.1 基本模块

AI 程序主要有以下几个模块：

- 初始布局
- 评分函数
- 着法的产生
- 搜索模式

2.2 初始布局

明棋规则下，布局对 AI 的水平有相当的影响。在布局的设计上，卢思迪助教在讲解大作业规则时提供了一个建议：随机生成布局，使用退火算法选择出较优的布局。但笔者发现，自己的 AI 在使用不同布局对战时有极大的概率平局，因此难以实现选择较优布局的过程。于是笔者采用的方式是：人工设计布局，并根据 AI 在具体对战中的表现通过人工分析对布局进行调整。



笔者采用的布局如上图。其中对地雷的安排采用了“三角雷”的模式，即将

三枚地雷放置在与军棋相邻的位置上，在明棋规则下，这种模式较为安全。

2.3 评分函数

评分函数的初步构思是，给予局面上的每个棋子一个分值，局面的评分即为所有棋子的分值之和。双方的同类棋子得分相同。

起初，笔者给每种棋子规定了一个固定的分值，但这种直接做法的缺陷是明显的，例如：当棋局上有司令但无军长时，师长已成为仅次于司令的棋子，此时仍给师长第 3 等的分值是不合理的。因此，笔者将评分方式更改为根据每种棋子在局面中现存棋子中的排名评分，这个改进带来的智能提升是可观的。

一些特定的处理：

- 军旗的分值设置为无穷大
- 若军棋被三枚地雷封锁了与外界的通路，则称这样的军旗是封闭的。给军旗的封闭性一个相当大的分值。
- 工兵的分值与对方军棋的封闭性有关，对方军棋封闭时工兵的分值较高。
- 旗上营（距军旗最近的行营）在棋局中往往有着很大的作用，尤其在三角雷的布阵中，旗上营控制了跨越旗上地雷调动棋子的通道。因此给予在旗上营的棋子一个额外分值。

在司令到排长一系列普通棋子的分值上，笔者的做法是，给不同排名棋子分值具有指数上的梯度。排名为 k 的棋子的分值约为排名为 $k+1$ 的棋子的分值的两倍。这样的分值梯度与笔者在五子棋 AI 的开发过程中使用的分值类似。这样的指数梯度使数个较劣棋子相加也难以超过较优棋子的估值，笔者在直觉上认为这是合适的。

在实际对战的表现中，笔者的 AI 常常能击败搜索层数更多的 AI，笔者认为这主要得益于评分函数的设计。

2.4 着法的产生

在搜索的每个节点（除最后一层节点），需要对当前的局面产生可能的着法，

由于军棋的规则和棋盘较为复杂，事实上这是整个 AI 程序中最繁琐的模块。

产生着法的方式是，枚举每个己方棋子，为其产生在各个方向上的着法。此时程序主要进行的是一个着法合法性的判定。为了简化程序中对棋盘不同位置的判定，笔者采用了较为便捷的写法。

```
//Normal-0 HeadQuarter-1 Camp-2 RailwayStation-3
int mark[H][W] = {
                    {0, 1, 0, 1, 0},
                    {3, 3, 3, 3, 3},
                    {3, 2, 0, 2, 3},
                    {3, 0, 2, 0, 3},
                    {3, 2, 0, 2, 3},
                    {3, 3, 3, 3, 3},
                    {3, 0, 3, 0, 3},
                    {0, 0, 0, 0, 0},
                    {3, 0, 3, 0, 3},
                    {0, 0, 0, 0, 0},
                    {3, 0, 3, 0, 3},
                    {3, 3, 3, 3, 3},
                    {3, 2, 0, 2, 3},
                    {3, 0, 2, 0, 3},
                    {3, 2, 0, 2, 3},
                    {3, 3, 3, 3, 3},
                    {0, 1, 0, 1, 0},
                };
```

如上图，在程序中加入一个常量二维数组，用不同的数值标记棋盘中的不同类位置（行营，铁路，大本营等）。另外，可以发现，一个位置的棋子可以沿某斜方向移动一步当且仅当此位置是行营或棋子沿此斜方向移动一步后到达行营。因此对于斜方向移动合法性的判断也可以很简单地实现。

依照“牺牲广度换取深度”的理念，为了增大搜索的深度，对所有合法的着法进行一定的筛选。删去所有“自杀”的着法，尽管这样的着法在一些极特殊情形下可能有意义。另外，由于工兵的合法着法极多，只保留其中的一些意义较大的着法，例如走向非空位置的着法和走向军棋相邻位置的着法。

2.5 搜索模式

采用“最大-最小”原理的搜索思想，搜索的过程即为产生着法，转移向下

一层节点的过程。一般而言，搜索层数越大，AI 做出的决策就更有远见。因此希望能够使搜索层数尽量大，具体优化措施将在第 3 部分“算法优化”中说明。

3 算法优化

3.1 Alpha-Beta 剪枝

Alpha-Beta 剪枝几乎是“最大-最小”搜索必备的优化，极易实现且能带来不错的效率提升。

Alpha-Beta 剪枝的思想是给每个搜索的节点一个分值范围 [Alpha, Beta]，对于一个己方节点（在其子节点中取最大值），不断用子节点的最大得分更新 Alpha。这样，在搜索此节点的子节点时，若找到一个分支到达分值小于 Alpha 的叶节点，可直接返回而不必搜索其他分支。

3.2 迭代加深的搜索

容易看出，若能先尝试得分高的决策，更容易在枚举之后的决策时产生剪枝。对于最大深度为 K 的搜索，可以先进行一次最大深度为 K - 1 的搜索，在根节点依照 K - 1 搜索得到的每个决策的得分将决策排序。

于是笔者采用了迭代加深的搜索方式。将最大搜索层数从 1 开始依次增加，直到达到时间限制。采用 clock() 函数获取当前运行时间，有助于尽量充分地利用有限的时间。这个优化将搜索层数提升至 5 层，且有一定比例的情形下能够搜索 6 层。

3.3 历史表

笔者此前采用的迭代加深搜索仅在根节点对决策排序，使用历史表后可在每个节点对决策排序，能够获得更佳的优化效果。历史表为一个数组，存储每种着

法的权值。最大深度为 K 的搜索使用的历史表由前 K - 1 次搜索得出。

在搜索的每个节点，增加此节点获取的最佳决策和产生剪枝的决策的权值，增加的数值为这个节点距叶节点距离的平方，这个数值的设定方式是由郑怜悯助教经实践得出的。

使用历史表后，搜索层数可以较为稳定地保持 6 层。=

3.4 随机化

使用上述算法的军棋 AI，在与特定 AI 的对战中经常出现平局的情况。为了尽量减少平局，可以在超过特定步数没有出现吃子时变着，选择次优的着法。但这样的解决方案存在一定风险，可能对己方不利。

使用随机化，也可以在一定程度上减少平局的概率。决策的随机化即为，在有多个决策分值相同时，随机选择其中的一个。另外，随机化也可以避免被某种固定着法重复击败。

3.5 其他优化

除了上述优化，后期对程序的优化主要是调整布局以及评分函数的常量参数。这些调整是通过人工分析对战的过程得出的，有时调整甚至使 AI 的水平变得更劣。这反映出笔者军棋水平的不足以及人工分析能力的局限。事实上，后期笔者花费的多数时间都用于对程序的参数进行反复调整。

3.6 部分优化日志

截至此时（11 月 27 日 01:30），笔者在评测网站上共提交了 100 个 AI 版本（用户名：SkyWalker）。其中后期的部分优化日志如下表：

编号	AI 名称	优化
90	交叉小径的花园	对布局进行了大幅度调整。

91	在每一条伤心的应天大街上	对估价函数的参数进行了较大调整。
92	黄昏现白骨	对布局进行了调整。
97	言叶之庭	加入了历史表。
100	你的名字	加入了决策随机化。

4 致谢

在 AI 的开发过程中，笔者参考了网站"象棋百科全书"中对于棋类 AI 算法的介绍。其中对各种算法优化方式的讲解使笔者受益颇多，尽管有些优化方式读者最终未能实现，但仍学习到了其中极具启发性的思想。

“象棋百科全书”网址：[http://www.xqbase.com/computer.htm]

感谢各位程序设计助教为此次军棋 AI 大作业付出的大量工作，例如：感谢高必成助教对评测机的数次故障修复以及后期增加多台评测机缓解高峰时段评测压力；感谢卢思迪助教开发的评测端程序以及 AI 对战可视化工具；感谢郑怜悯助教在习题课上讲解 AI 调试技巧，以及他参与到 AI 对战中促进了各位同学的 AI 开发进程。

感谢在 AI 开发过程中与笔者进行诸多交流与讨论的林虹灏、刘啸远、冯思远等同学，在与他们的交流过程中，笔者接触了不同的 AI 设计思路，受到了有益启发。尤其感谢林虹灏同学向笔者分享他使用历史表等优化方式的实践经验。