Report for Holdem AI

陈子豪[1]

Augest 2014

[1]5130309305 上海交通大学 2013 级 ACM 班

Contents

Chapter 1

导言

德州扑克是一种有趣的扑克游戏，德州扑克的游戏过程中，能否取胜很大程度上取决于玩家对概率的计算或者直观感觉。

在前几次 AI 的测评的时候，The Joker 展现出了较为强大的战斗力，The Joker 是一个理性主义者，它的决策基于对期望的计算，其强大战斗力背后是数学在发挥核心作用。

本文对作者的 AI 的策略以及实现进行详细介绍。主要有：利用蒙特卡洛方法估计牌力，数学期望计算进行决策，以及作者自己对“计算最小加注”、“showdown”函数的实现。

本文将先给出 AI 的框架，再详细叙述一些较为复杂的“计算最小加注”、“showdown”函数的事项，之后将详细叙述自己的决策方法和理由，最后叙述自己在 AI 构思与实现过程中的一些体会。

Chapter 2

AI 框架

AI(The Joker) 分为 Player.h 以及 Player.cpp 两个文件，其中 Player.h 文件为 Player 类的声明，Player.cpp 文件为对类中函数的实现。

Player 类中包括以下若干个函数：

- int findCurrentBet();

  计算出当前所需跟注大小

- double calHandStrength();

  计算牌力

- int minRaise();

  计算出当前加注所需家的最小注

- double potOdds(int);

  计算出底池赔率 potOdds = (自己在池中的总筹码数 + 自己将要投入的筹码数)/(池中总筹码数 + 自己将要投入的筹码数)

- bool randomGenerator(int percent);

  以 percent% 的概率返回 true，否则返回 false

- decision_type FCR();

  FCR 表示 Flop，Call，Raise，用于对当前情况进行判断，返回一个决策。

函数返回只有三种决策：Check，Call，Raise，由于 Fold 与 Check 都是不需要再加入筹码的，因此只要返回 Check 即可，一些细节（比如在 PreFlop 阶段大盲注有可能需要 Call 0）并不需要在本函数中考虑，在各个阶段的决策函数中（即 preflop(),flop() 等）进行特判即可

- int convert(char);

  用于将’2’...’T’...’A’ 转换成数字

- string generateString(const vector<card_type> &);

  用于找出 7 张牌中最大的 5 张组合时所要用到的函数之一，将一组 5 张的牌转换成字符串进行比较

- int better(const vector<card_type> &, const vector<card_type> &);

  用于找出 7 张牌中最大的 5 张组合时所要用到的函数之一，比较两个组合大小

- void dfs(int, int, const vector<card_type> &, vector<card_type> &, vector<card_type> &);

  用于找出 7 张牌中最大的 5 张组合时所要用到的函数之一，搜索得到 7 张牌中的所有牌型组合

- decision_type preflop();

  在 preflop 阶段决策

- decision_type flop();

  在 flop 阶段决策

- decision_type turn();

  在 turn 阶段决策

- decision_type river();

  在 river 阶段决策

- hand_type showdown();

  进行 showdown

Chapter 3

复杂函数实现

3.1 showdown 函数的实现

showdown 函数考察的是基本功。大体思想是这样的：进行一次深度优先搜索从 7 章牌中得出所有的 5 张牌的组合方案，接下来进行比较，记录最大的牌型，最后并返回。

复杂的地方在于牌型的比较，主要方法是：从小到大给不同等级的牌一个编号，等级越高编号越大，High Card 到 Straight Flush 依次为 0 - 8，(Royal Straight Flush 可以包含在 Straight Flush 中)。对于一组 5 张的牌，先判断其牌型并得到一个 0-8 之间的编号，接下来会将其转换成一个字符串，字符串的第 0 位便是刚刚所得到的编号，接下来对这五张牌的大小从大到小进行排序，之后保证四条、三条、对子尽量靠前，如牌 A2222，要变成 2222A, 牌 55444，要变成 44555，这样便可以得到一个长度为 6 的字符串。接下来的比较仅仅需要对字符串进行比较即可。

不同函数之间的逻辑调用关系是：showdown() 调用 dfs()，dfs() 调用 better()，better() 调用 generateString()。(函数的参数在这里省略了)

如上所述，实现过程中作者用了大量的函数，进行了相对比较高度的封装。这样做有两个原因，第一个是可以使实现的时候逻辑更加清晰、不易出错，另一个原因是在决策时概率计算的时候会用到其中的一些函数，这样做增强了代码可重用性。

3.2 求出当前最小加注量的实现

对于如何求出当前最小加注量，我写之前大家都没有简单的写法，这里我给出一个简短的写法。

```
int Player::minRaise()
{
    vector<int> bets = query.current_bets();
    int numOfPlayers = query.number_of_participants();
    int cur = myId;
    int gap = 0, tmpMax = bets[cur] ;

    for (int i = 1; i < numOfPlayers - 1; i++)
    {
        cur = (myId + i) % numOfPlayers;
        if (query.player_status(cur) == BET && bets[cur] > tmpMax)
            gap = bets[cur] - tmpMax;
        tmpMax = max(tmpMax, bets[cur]);
    }
    if (gap == 0) gap = 2 * query.blind();
    return gap;
}
```

主要用到的是 curren_bets() 这个 vector，这段代码的大致思想是这样的，找出自己当前如果要 Call 所需跟的注是比较容易的，只要将当前轮玩家投入最多的注减去自己投的注即可。事实上，此时如果要求之前每个玩家在他当时进行决策的时候如果 Call 所要加的注也是比较容易的，也可以用同样的方法得到。

因此，在上面一段代码中，会从自己开始往后扫一圈，每次记录扫到现在所得到的玩家的最大投注量。同时也需要考虑有玩家 allin 或者 fold 的情况，因此会有判断“bets[cur] > tmpMax”，如果扫了一圈所有人都是一样的话，代表本轮还没有人加过注，而最小加注为一个大盲注，便返回一个大盲注。

有很多同学抱怨说接口不够友好，算最小 raise 比较麻烦，但如果静下心来想一想投注序列的性质的话，也不难想出上面给出的这种简便的做法。

Chapter 4

决策

AI The Joker 的决策取决于自己所已知的牌，不受其余玩家决策的影响。会对自己进行加注、跟注后赚得钱的数学期望进行计算，如果期望为负则跟注或者弃牌。

下面给出期望正负的判定方法。大致思想是，如果接下来要判定加注后的期望，那么首先先算出**底池赔率**，底池赔率

$potOdds =$ (自己在池中的总筹码数+自己将要投入的筹码数)/(池中总筹码数+自己将要投入的筹码数）

接下来根据自己所已知的牌算出自己获胜的概率**手牌力量** HS(Hand Strength)，如果 $RR = HS/potOdds > 0$，那么期望为正，否则为负。

细心的读者会注意到一个问题。

一方面，在计算获胜概率的时候，所选取的玩家人数是轮到自己决策的时候没有 Flod 的人数，然而，自己并不知道在自己之后有多少人 Fold，于是，这样算出来的低于实际的概率。

另一方面，在计算底池赔率的时候，也会出现两个问题，第一个问题，自己目前在彩池中的总筹码数是容易求得而且是没有什么大问题的，然而，自己将要加的注就不能只考虑 Call 或者 Raise 的情况下的 HS 的大小，要分别算出两种情况下的 HS 的值。第一个问题是比较容易解决的，除此之外，还有第二个问题。在计算底池赔率的时候，自己也不知道在自己决策之后，后面的人会进行怎样的加注。然而，后面人的加注情况会影响底池赔率的大小，同时也会影响我们希望得到的数学期望的大小。

这两个方面的问题，都是比较棘手的问题。一个可行的想法是，对所有对手的打牌方式进行记录与计算分析，然后估算出自己决策后生下来的玩家的 Fold 比率以及平均投注额。但是，这个解决方案成本很高，而且由于测试的时候次数比较少，这种解决方

法甚至有可能产生负面的影响。最后我采取的解决方案是，在负责决策的几个函数中利用随机概率修正。这里需要提的是，作者在进行决策并不是严格根据期望（比如期望为负的时候就一定不 Call 或者 Raise），而会利用随机来进行最终决策，不同决策获得的随机概率，这样就会得到当前情况下，每种决策的分布，而作者进行了调整之后使得分布最终受赢钱期望决定。这么做的原因有两个，第一个是在解决刚刚提到的两个方面的问题的时候，可以通过一定的参数修正来解决那两个问题。第二个原因是有的时候，可以起到诈唬的效果，事实证明，诈唬还是可以带来一定的收益的。

接下来给出计算 HS 的方法，最为暴力的方法是枚举出所有情况，将剩余公共牌与其他玩家的牌型全部都枚举出来，计算出自己的牌最大的情况数，再除以总情况数即为获胜概率。但是显然，这并不是一种非常好的方法，因为可能的情况数很多，多的时候 10 秒也难以跑完，那么一个可行的方法是进行蒙特卡洛模拟，模拟的过程如下：

```
preparations

score = 0

repeat 2000 times

    shuffle the remaining cards

    if (my card is the best) score++

    else add 1/(number of people having the same strength card with me)

end repeat

hand strength = score / 2000
```

实验表明，模拟 2000 次已经能使概率非常精确，对比网上的概率表，这种模拟方法的最大偏差不超过 2%，这已经完全可以为 AI 的决策进行服务了。模拟过程中很重要的一点就是不要忘记对平局的处理，上述模拟过程也考虑到了对平局的处理。

除此之外，在自己手中筹码不多的时候，AI 会比较谨慎，我在这里加入了一个判断来对剩余筹码进行保护。

```
if ((myRemainingTokens − currentBet < query.blind() * 4) && HS < 0.5)
    return make_decision(CHECK);
```

同样者的一提的是，前期由于盲注较小，如果遇到好牌加注的时候只加最小要加注的值过于保守，因此我会对前期的加注大小进项上调。根据一场比赛的若干阶段，前期

中期后期也会采用不同的决策倾向，前期比较凶，有好牌就一直 Call 或者 Raise，中期有的时候会有一些诈唬，后期的策略相对较稳，但是由于盲注的提高，后期 AI 将会有较大的发力，真正赢钱还是在后期。The Joker 在实战中，showdown 的胜率比较高而且次数也并不少。

Chapter 5

感悟

写一个这样的 AI 还是一件十分有趣的事情，自己也是首次写蒙特卡洛模拟，看到得出结果比较精确还是很有成就感的。总的来说，我是比较满意我的 AI 的表现的。

AI The Joker 是一个比较理性、决策基于概率、各阶段倾向不同的一个 AI，理论上应该是一个有趣而且有一定战斗力的 AI。