A Monte-Carlo Agent for No-Limit Multiplayer Texas Hold'em Based on Economic Value Added(EVA)

Design Report

University of Chinese Academy of Sciences
Hongyan XUE(薛弘彦), Jingpei HU(胡景佩), Weihao KONG(孔维浩), Yuncong GAO(高云聪)

0. Table of Contents

Table of Contents

1. Abstract

In this project, we developed an intelligent agent for no-limit multiplayer Texas Hold'em Poker. The agent adopts a EVA(Economic Value Added) based strategy, where the agent chooses the action with the maximum expected economic reward. The expected reward is computed using a Monte-Carlo method.

The agent has been tested in a multiplayer environment and has shown promising performance.

2. Algorithm Overview

The project is implemented in C++ for performance considerations.

It also features a Python interface so that it can be tested in a Python benchmark, where it could be benchmarked against multiple other agents.

The algorithm consists of two parts.

I. An evaluation algorithm, where the agent computes its winning probability against each opponent.

II. A decision algorithm, where the agents makes the move that maximizes its EVA (Economic Value Added, same below) value. However, other factors are also taken into account in order to limit risk.

3. Decision Algorithm

The decision algorithm executes in the following manner:

I. It computes the probability that its hand is better than every other agent.

II. For each possible action:

a) Compute the EVA of this action. In other words, the expected profit of this action minus the expected investment.

b) If this action is better than all other actions, mark it as the best action.

III. Return the best action. If there is none, return “check” or “fold”, depending on the circumstances.

3.1.1. Winning probability

The probability of being better than every other agent is:

$$P_{AllBetter} = \prod_{for\ each\ opponent} P_{BetterThan(opponent)}$$

The method of computing $P_{BetterThan(opponent)}$ will be shown in section 4.

3.1.2. EVA value

The EVA value of an action is:

$$P_{AllBetter} \times PoolValue - (1 - P_{AllBetter}) \times Investment(action)$$

Where $1 - P_{AllBetter}$ is the probability of losing the investment of this “call” or “raise”.

The $PoolValue$ is an estimate of the future pool after a round of betting.

It is computed assuming that, after a round of betting, every other player will call, and make the

There is a limited amount of legal actions for this AI. The investment of each action is shown below:

```
int amount(Action act,Observation* obs){
    int minAmount=0;
    for(int i=0;i<obs->numPlayers;i++){
        minAmount=max(minAmount,obs->stages[obs->currStage].contribution[i]);
    }
    if(act==CALL){
        return minAmount;
    }else if(act==RAISE_3BB){
        return minAmount+3*obs->bigBlind;
    }else if(act==RAISE_POT){
        return obs->communityPot+obs->roundPot;
    }else if(act==RAISE_2POT){
        return 2*(obs->communityPot+obs->roundPot);
    }else if(act==ALL_IN){
        return obs->stack[obs->position];
    }else{
        return 0;
    }
}
```

Figure 1. Investment of Each Legal Action

Note that, this AI completely ignores the previous investments in other rounds.

This makes sense economically, since the previous investments cannot be recovered. The only

utility of the previous investments is the fact that it kept the agent in the fight, and that it might have raised the value of the pot.

This is known as the Sunk Cost in the field of economics.

4. Evaluation Algorithm

The evaluation algorithm estimates the probability that the agent has a better hand than an opponent.

The evaluation algorithm is, for now, purely based on the card combination. It evaluates the winning odds by purely looking at the current hand and the community cards.

In the future, more sophisticated algorithms can be added, such as those that are based on the actions of other agents.

4.1. Evaluation Overview

An evaluation has the following steps.

The "rank" of seven cards is the probability that the best subset of 5 cards is better than a purely random set of 5 cards.

I. For the agent itself, enumerate all possible combination of the rest of the community cards in a depth-first manner.
   For each such enumeration, the set of 7 cards of the agent would be fixed.
   Next,
   a) Compute the rank of the agent in this enumeration, using the community cards and the known hand cards.
   b) Record this rank in a list.

II. For the opponent, randomly generate the rest of the community cards and the 2 hand cards. Generate 1000 such Monte-Carlo samples.
   For each sample, the set of 7 cards of the opponent would be fixed.
   a) Compute the rank of the opponent in this enumeration, using the community cards and the random hand cards.
   b) Record this rank in another list.

III. For each enumeration in I,
   a) Compute the probability that the rank of the agent is high than that of the opponent. This is done by finding the percentage of the 1000 samples that are worse than the agent's hand.
   b) Average the ranking over all enumerations.

We would then acquire an estimate of the probability that the agent has a better hand than an opponent.

The reason why enumeration is used in I, while Monte-Carlo sampling is used in II, is discussed in 4.3.

4.2. Rank

The "rank" of a set of card is a measure of how good the set is. We only define the "rank" of

seven cards.

In order to calculate the rank of seven cards, we must enumerate every possible C_7^5 subset of 5 cards.

For each subset, we give a probability estimate that it is a better hand than a purely random set of 5 cards.

Finally, we find the maximum probability amongst the C_7^5 combinations.

4.2.1. Ranking algorithm from SKPokerEval

There is a known project on Github, known as the SKPokerEval, which conveniently provides a function that evaluate seven cards.

The repository can be found at https://github.com/kennethshackleton/SKPokerEval

4.2.2. Authentic ranking algorithm

We have also developed an authentic ranking algorithm.

We would not go into the technical details of this algorithm. In essence, it checks if the given 7 cards form any of the hands in the ruls of Texas Hold'em, and gives a rank accordingly.

A documentation containing the details of this algorithm can be found in the appendix.

4.3. Monte-Carlo vs Enumeration

There are two ways to generate the unknown cards.

I. We could adopt a Monte-Carlo method and generate them randomly, and then compute the average.

II. We could enumerate all the possible hands and compute the average.

In practice, both methods have been attempted.

We have found that it is impossible to adopt method II in the generation of the opponent's hand. There are just too many possible combinations. Therefore, method I is used for the opponent.

On the other hand, method II is acceptable for the agent. This is because there are far less combinations to explore. Therefore, method II is used for the agent itself for accuracy.

5. Contributions

Hongyan XUE（薛泓彦）: C++ to Python interfacing, and the decision algorithm.

Jingpei HU（胡景佩）: Authentic hand evaluation algorithm.

Weihao KONG（孔维浩）：Documentation and the decision algorithm.

Yuncong GAO（高云聪）：Algorithm optimization and research of current algorithms.

6. Appendix

6.1. Authentic Evaluation Algorithm

A Chinese version of the documentation is listed below.

我们考虑每种初始牌型，在我们看来，无非就是牌的大小以及是否同色，一共 13*12/2（同色 C(13,2)），13*13（不同色）种情况，但实际上，考虑全体组合，应该为 C(52,2)，主要在于花色有多种选择，实际全部枚举后划分为 169 种情况。

考虑手牌固定的情况下，枚举剩下的五张公共牌，通过五张牌的全部组合，我们可以计算出每次初始牌对应的期望，由于低分过于集中，暂时决定在所有情况中进行排序比出相对胜率，但是这显然存在问题，由于当前的手牌也会影响到别人手牌获取的概率，需要做贝叶斯处理，但是考虑到仅仅计算初始牌期望，每个人的条件是等同的，简化计算可以被接受。

简化计算的话，为了尽可能降低复杂度，我们同时枚举出 7 张随机牌，算出这张牌的最大分数，之后，7 张牌中选出所有两张牌的情况作为基底牌，通次分数累加值与累加次数，得出该基本牌的期望值，然后将期望进行打表，表示 updata.csv，在调用函数的时候进行临时的读入。

之后以上述期望和概率表作为参照，为后续河牌等环节提供参考。

我们运行每次提供

1. 手牌 的优势概率
2. 手牌+3 张明牌的优势概率
3. 手牌+4 张明牌的优势概率
4. 手牌+5 张明牌的优势概率
5. 大于等于 5 张牌时的分数

存在一定问题，进行预测时，数据的分布过于集中，导致概率变化的不明显，暂时采用简单的分段手段，在分数较低的预测值进行概率测试时，有不错的预估性。

dfs 函数左右深度遍历，进行期望值合理性的前期检测。

rd 函数作为读入函数，可以调整为 agent 接口，运行做任何修改。

deal 函数用于先期的预处理，即生成表的函数，由于我们事先生成好了，在后语过程中该函数不需要再次被调用，只需要读取打好的表。

table 函数，将预先打好的表读入进来，以及期望和优势概率的比照。

main 函数，读入的接口。

calcu 函数，用于计算 5 张即 5 张牌以上的最佳分数，分数的关系在映射表中给出，或者查看牌型具体的函数。

is_royal_flush…表示是否为皇家同花顺。

其他的”is…”函数的定义类似。