Finding the Best Algorithm for Playing "Dueling Fool"

Daniil Leskevich
Optimization Class Project. MIPT

Introduction

Card games have always attracted mathematicians because of the random distribution of cards and incomplete information from the players.

"Fool" is a traditional card game popular in the countries of the former USSR. The goal of the game is to get rid of all the cards.

In my project I create state evaluation functions that depends on heuristic, and try to find best combination of this heuristic

In math language we can say $min - f(x, S)$, where S all combination of x, and $f(x, S)$ return win rate x on this space.

Hand Cost

Heuristics are various methods for estimating a state. The state is evaluated only at the first stage of the game, because when the game enters the game phase with complete information, the correct combination of moves is selected by brute force.

In this work, the following formula will be used to evaluate your hand

$$\text{cost} = \text{CardCost(C)} + \text{PairCost(P)} + \text{rankSpread(R)} + \text{scatterBySuit(S)} + \text{trumpBonus(B)}$$

, where CardCoast is the calculated weight of all cards, PairCoast is the calculated weight of all paired cards, rankSpread is the rank spread, scatterBySuit - suit scatter, trumpBonus - bonus for trump cards in hand.

In this case our task is to find the best set of these coefficients C, P, R, S, B

How we attack

Let's describe what bots do in attack.

- decide whether toss a card and which one
  - Based on known information about deck estimate average cost of card from the deck(Use CardCost(C) and trumpBonus(B))
  - Multiply our average deck cost by coefficients with depend on (last card, number of cards in deck and heuristics params)
  - Find an array of probabilities , in which for each card the probability of throwing another card is stored if the opponent is beating off
  - For all options, we find the new value of our hand and choose the option with the highest expected value of the hand.
    New hand cost = (Hand without card cost)+ (probabilities to add)*(heuristics param) * (average cost of card)
  - Compare Hand cost with cost if we don't add card and choose best
  - Return card for attack or flag that we don't want to continue attack
- Opponent take cards from table and we should decide want cards we need to toss him
  - all items from the case "decide whether toss a card and which one" which we repeat "how many card we can toss" times

How we defence

Let's describe what bots do in defence.

- Based on known information about deck estimate average cost of card from the deck(Use CardCost(C) and trumpBonus(B))
- Multiply our average deck cost by coefficients with depend on (last card, number of cards in deck and heuristics params)
- Find an array of probabilities , in which for each card the probability that opponent throw another card.
- For all options, we find the new value of our hand and choose the option with the highest expected value of the hand.
  New hand cost = (1+(Card in opponent hand)*(heuristics param))*probabilities)*(Hand without card cost) + (average cost of card)
- Compare Hand cost with cost if we don't defence (heuristics param)*cost(Hand+Cards on table) and choose best
- Return card for defence or flag that we don't want to continue defence and take all cards.

Algorithm

Simplified view of optfunction

```
GET (x,BotsForCompare,NewBots)
Initialize winRate = 0
for bot in BotsForCompare do
    for i = 0, 1..n − 1 do
        winRate += playGame(bot, x)
    end for
end for
if winRate > min(winRate(NewBots)) then
    NewBot.remove(min winRate element)
    NewBot.append(x, winRate)
end if
Return winRate
```

Simplified view of study

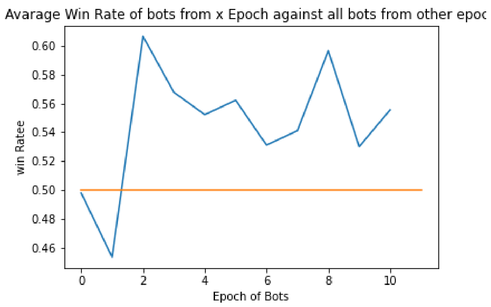
```
Initialize BotsForCompare
Initialize BotsFromPrevEpoch
for i = 0, 1, .., n − 1 do
    Initialize NewBots = emptyList()
    differentialEvolution(optfunction, args = (BotsForCompare, NewBots))
    NewBots = compare(BotsForCompare, BotsFromPrevEpoch, NewBots)

    BotsFromPrevEpoch = BotsForCompare
    BotsForCompare = NewBots
end for
```

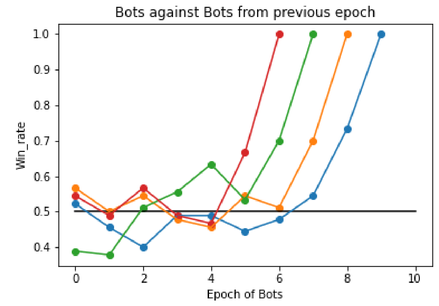
How we study, momentum matrix

After we find matrix B_k with best heuristics against bots from matrix A_K, we say that $A_{k+1} = compare(B_k, A_k, A_{k-1})$, compare return bots with highest win rate against bots from matrix $[B_k, A_k, A_{k-1}]^\top$. A_{k-1} we can call **momentum matrix**, because it help select bots with highest generalizing ability.

Numerical example



Results



In my project i'm find bot with the best Heuristics. To find this best Heuristics i will compare average bots winrate on the list of bots.

Code of project https://github.com/Lisenok666/OPT_DURAK

Conclusion

we can find a winning strategy only for a certain function $f(x, A_n)$ $A \in S$ and when changing the matrix A_n x will not be the optimal solution even for $S = \cup_{i=0}^{i=n} A_i$

References

[1] AI for "Fool" (2015) *https://habr.com/ru/post/263259/*.

[2] Algorithms and programs for computer control in gambling, created on the basis of fuzzy set theory (2009) *https://inlnk.ru/G6YdlO*, A. F. Lyakhov, I. V. Trushin

[3] Information analysis of gambling(2006) *http://m.mathnet.ru/php/archive.phtml?wshow=paper&jrnid=mo&paperid=445&option_lang=rus*, A. F. Lyakhov.