Using Monte Carlo Tree Search on the card game Briškola

Artificial Intelligence 2016/17, Faculty of Computer and Information Science, University of Ljubljana

Boris Karamatić

***Abstract*—When we are developing a game AI there are many different methods we can use. One of those is Monte Carlo Tree Search which allows us to develop the states tree for as much time as we have available and then return the most promising move. In the following report we will present our solution for the problem of developing an AI capable of playing the card game Briškola, for which we will use the Monte Carlo Tree Search.**

I. Introduction

When developing an AI capable of playing games, the AI has to be convincing. It has to trick the player into believing that the AI he is playing against is smart, even though it most often is not. In the following report we will describe our solution for developing an AI capable of playing the card game Briškola without cheating. First we will present the game Briškola and its rules, then we will describe the method we used, that is the Monte Carlo Tree Search (MCTS). After that we will present the details of our implementation and at the end we will present the achieved results and give our concluding thoughts.

A. Briškola

Briškola is a card game which is played with 40 cards of 4 different colors, with 10 cards per color. The cards can be seen in figure 1. Cards go in strength from strongest to weakest in the following order: 1, 3, 13, 12, 11, 7, 6, 5, 4, 2. Cards 2, 4, 5, 6 and 7 are worth no points, 11 is worth 2 points, 12 is worth 3 points, 13 is worth 4 points, 3 is worth 10 points and 1 is worth 11 points. In total there are 120 points and the goal of the game is to collect as many points as possible. The player which collects over 60 points wins the game. In case that both players collect 60 points, the game ends in a draw.

At the start of the game the deck of cards is shuffled and each player is dealt 3 cards. One card is than put under the deck facing up and the color of that card becomes the briškola of the game. The cards of color briškola can collect all other cards in the game, except for higher briškolas. The player who is going first drops a card, if it is of color briškola, then the second player can collect it only with a higher briškola. If it is of any other color, then the second player can collect it with either a higher card of the same color or with any briškola. Who ever collects the cards is playing first the next turn. In order of play each player then first draws a card and after that in order of play each player drops a card. This continues until all cards in the game have been played.

B. Monte Carlo Tree Search

The basis of MCTS is the simulation of games where both the AI-controlled player and its opponent play random moves, or even better, pseudo-random moves. Not much can be learnt from a single random game. But from simulating many random games, a good strategy can be inferred and this is what MCTS abuses [2]. MCTS uses the following four steps to build a tree. The steps are Selection, Expansion, Simulation and Backpropagation. The outline of MCTS can be seen in figure 2.

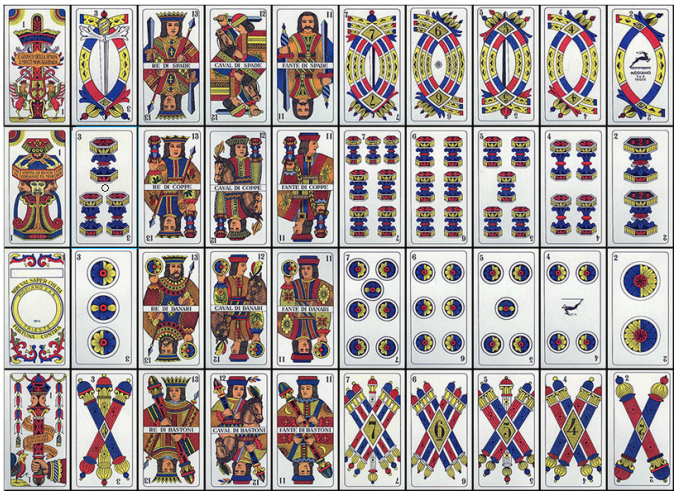

Figure 1. The cards used to play Briškola. Image taken from [1].

Selection In the first step *Selection*, the algorithm selects the best child from the root until it reaches a leaf node or a node that has not been yet fully expanded. The best child is selected according to a formula that balances between exploitation and exploration. On one hand we want to select and develop the game actions that lead to the best results so far (exploitation) and on the other hand we want to explore the less promising actions, due to the uncertainty of evaluation (exploration) [2]. If the leaf node has not yet been visited, we skip the expansion and move to the simulation. If the node has already been visited we continue with the expansion.

Expansion After we have selected a node that has not been yet fully expanded, we expand it by adding new nodes from the list of valid moves. After that we move into one of the newly added nodes.

Simulation From this node we run a game untill it ends using random moves. If all legal moves have the same probability of being selected than of course the strategy of play is going to be weak, and the level of play of the Monte Carlo program suboptimal. There for using heuristic knowledge to give larger weights to actions that are more promising is advised [2].

Backpropagation After finishing the simulation we need to update each node back to the root, increasing the number of visits for those nodes and updating their win/loss ratio.

After MCTS runs out of time, the game action that is executed is the one corresponding to the child which was explored the most [2].

II. Methodology

When developing an AI which is capable of playing Briškola, if we do not want our AI to cheat by knowing the cards in our opponents hand or by knowing how the deck is shuffled we face the problem of running out of time before we are able

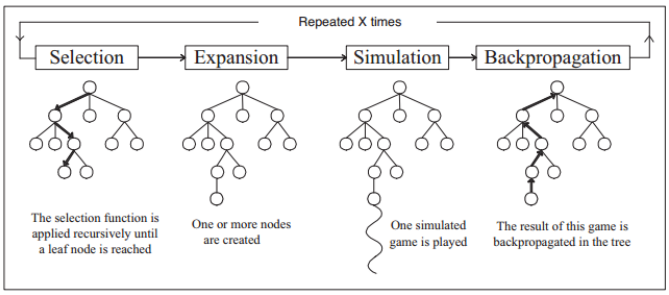

Figure 2. Outline of Monte Carlo Tree Search. Image taken from [2].

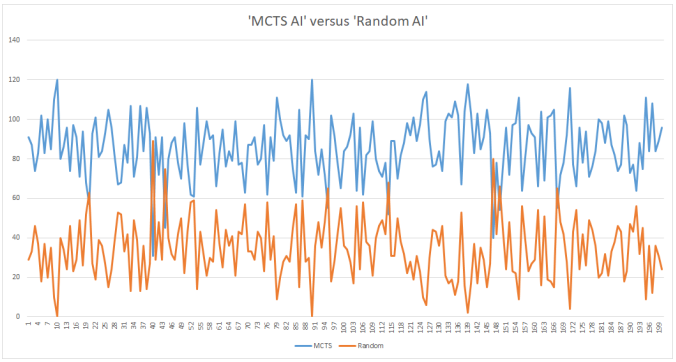

Figure 3. The achieved result of our AI after playing 200 games against an AI that was doing random moves.

to simulate all possible moves. Lets say for example that it is the first turn of the game and we are going first. Which of the three cards in our hand do we drop? For each card we drop, our opponent can drop any of the remaining unknown 36 cards. After that when we draw a card, we can draw any of the remaining 35 cards which then in result causes 37 possibilities for what we could drop the next turn. The tree can quickly grow out of control and we do not have the time to develop the whole tree for each play.

That is why we decided to use Monte Carlo Tree Search (MTCS) which has the ability to keep developing the state tree for as long as we allow it and then return the most promising play. Another benefit of using MTCS is that it is capable of recognizing which play is the most promising and spend more resources on researching that branch. And while doing that it won't ignore the other branches, it will just spend more time in the most promising one. Another thing is that the simulation part of MCTS can be run with completely random moves, which means we do not have to develop any heuristic for which move is the best one.

We decided to implent our solution in Python and in doing so we helped ourselfs with [3].

We built two artificial intelligences to play Briškola. One that uses only random moves and the other that uses MCTS to find the optimal move. The last one had a second each turn to find the best move. When choosing the best child we used the following function:

$$score = \frac{child.reward}{child.visits} + \frac{1}{\sqrt{2}}\sqrt{\frac{2 * \ln(node.visits)}{child.visits}} \quad (1)$$

In case that the *child.visits* was equal to 0, the score was set to $Infinity$. At the end of simulation when we returned a reward to be backpropagated we returned the achieved amount of points divided by 60. This way we not only rewarded the plays that lead to victory, but plays that lead to the maximum amount of points.

When we were expanding a node we could have expanded it with one node at a time or with all the nodes with all possible plays at once. We decided to do the later. In the simulation part of the algorithm we used completely random moves.

You can find our implementation on github at [4].

III. Results

We tested our AI by making it play 200 games against an AI that was making random plays each turn. Our AI had one second of time to find the best play and after 100 games we achieved a win rate of 96%. The results of all games can be seen in graph 3.

The achieved results seem quite promising. But when we analyse the plays by ourself we can sometimes notice that the AI is capable of doing some questionable plays. For example when going first it often likes to drop a high value card. We contribute this to the simulation part of MCTS where we used completely random moves, resulting in our AI believing that dropping a high value card is hard to punish and there for going for that move. Another problem with our AI is that it prefers going first which means it will often collect opponents non value cards even though it did not have to do it. Again we contribute this to the completely random simulations where going first is better since it has a high chance of going unpunished by a random move. But from playing the game we know that going second is better since reacting to our opponents moves is better then dropping a blind card in risking that our opponent collects the card with a high value non briškola card.

IV. Conclusion

Our solution compared to a solution that plays only random moves achieves good results. But there is definitely room for improvement. In our simulations instead of using completely random moves we could have used pseudo-random moves. That way our AI could learn to do better, more realistic moves. We could also give him more time to think each move, which should let him explore the possible plays even more, yielding even better results. We could also try to use some other function to decide when to explore and when to exploit.

As it stands right now our AI performs really well against an opponent that makes random moves. But if it had to play against a human opponent it could get heavily punished for dropping a high value card when going first.

Developing AI for games is a fun challenge and MCTS proves to be a great solution. MCTS was also used by Google DeepMind to develop an AI that was for the first time capable of beating a professional Go player at the board game Go [5]. Of course their solution did not simulate the games using completely random moves instead they used neural networks in the simulation part. But non the less this shows the power of MCTS and its viability in developing strong game artificial intelligences.

References

[1] Wikipedia, "Briškola — wikipedia, the free encyclopedia," 2013, [Online; accessed 18-June-2017]. [Online]. Available: https://sl.wikipedia.org/wiki/Bri%C5%A1kola

[2] G. Chaslot, S. Bakkes, I. Szita, and P. Spronck, "Monte-carlo tree search: A new framework for game ai." in *AIIDE*, 2008.

[3] H. Sultan, "Mcts," https://github.com/haroldsultan/MCTS, 2015.

[4] B. Karamatić, "Mcts-briskola," https://github.com/K-Boris/MCTS-briskola, 2017.

[5] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.