CE888-7 Data Science and Decision Making

Submitted as part of the requirements for:

CE888-7 Data Science and Decision Making Assignment 1

Name: Wenbo Bao

Tutor: Ana Matran-Fernandez

Date: 08 February 2019

Table of Contents

1. Abstract

In this report, I focus on studying Monte Carlo Tree Search (MCTS), which is a kind of decision trees for expert iteration (Reinforcement learning), through generating and collecting the data of multiple games, like OXO, Othello and Nim. This report presents an analysis method. Through this, we can achieve the knowledge about what is MCTS, how to apply it in different area and what is the advantage and drawback of MCTS.

Index Terms—Monte Carlo Tree Search (MCTS), Upper Confidence Bounds for Trees (UCT)

2.Introduction

Studying MCTS that is a kind of decision trees for expert iteration (Reinforcement learning). Using the provide code, generate and collect combinations of state values data and the action that the agent took data for at least one kind of games (e.g. OXO, Othello and Nim). During the procedure of this method process, I have learnt more details about MCTS and find more relevant information that I need to study. This report outlines the core algorithm's derivation and the underlying mathematics behind MCTS. Besides, as obtaining the complete sample data, which can be fed for any other machine learning algorithm, could help us to improve the performance of present algorithm.

3.Background

Monte Carlo tree search (MCTS) is consisted of decision theory, game theory, and Monte Carlo and bandit-based methods.

MCTS is a proposed search method that combines the precision of tree search with the generality of random sampling. It not only has received considerable interest due to its spectacular success in the difficult problem of computer Go, but also has proved beneficial in a range of other domains. [1]

In the five years since MCTS was first described, it has become the focus of much AI research. Spurred on by some prolific achievements in the challenging task of computer Go,

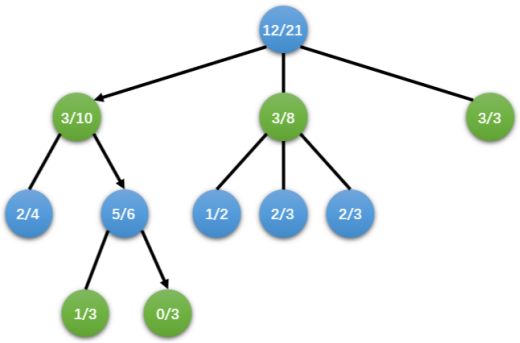
researchers are now in the process of attaining a better understanding of when and why MCTS succeeds and fails, and of extending and refining the basic algorithm. These developments are greatly increasing the range of games and other decision applications for which MCTS is a tool of choice and pushing its performance to ever higher levels. MCTS has many attractions: it is a statistical anytime algorithm for which more computing power generally leads to better performance. It can be used with little or no domain knowledge and has succeeded on difficult problems where other techniques have failed. [2]

Monte Carlo Tree Search (MCTS)

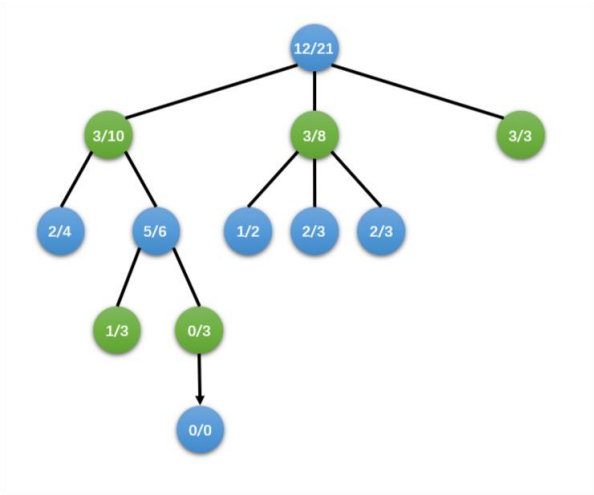
The basic Monte Carlo Tree Search (MCTS) process is a simple algorithm. A tree is built in an incremental and asymmetric manner. For each iteration of the algorithm, a tree policy is used to find the most urgent node of the current tree. The tree policy attempts to balance considerations of exploration (look in areas that have not been well sampled yet) and exploitation (look in areas which appear to be promising). A simulation is then run from the selected node and the search tree updated according to the result. This involves the addition of a child node corresponding to the action taken from the selected node, and an update of the statistics of its ancestors. [2] [3] [4]

Each round of Monte Carlo tree search consists of four steps: [2]

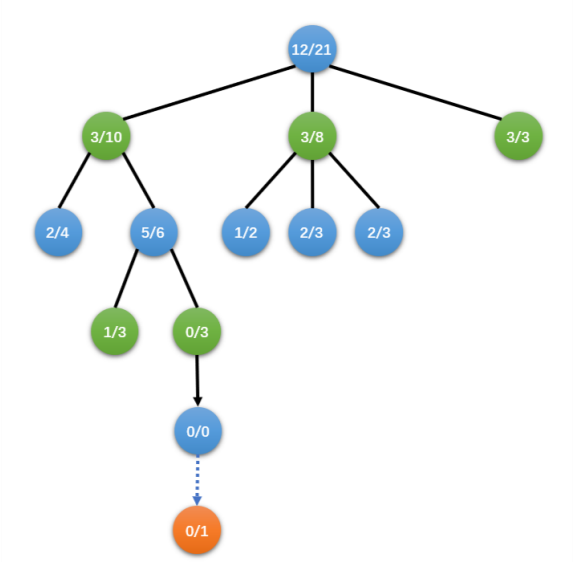
1) Selection: Starting at the root node, a child selection policy is recursively applied to descend through the tree until the most urgent expandable node is reached. A node is expandable if it represents a nonterminal state and has unvisited (i.e. unexpanded) children.



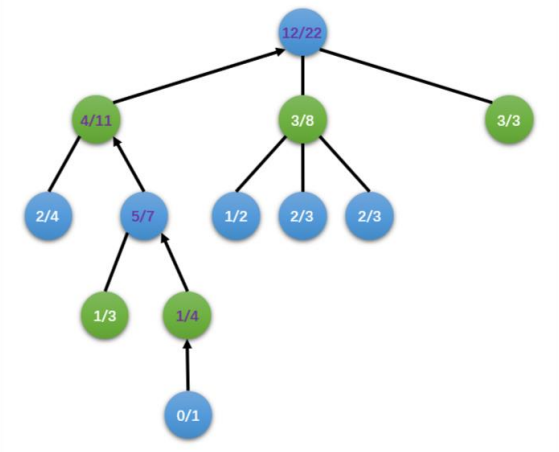
2) Expansion: One (or more) child nodes are added to expand the tree, according to the available actions.



3) Simulation: A simulation is run from the new node(s) according to the default policy to produce an outcome. This step is sometimes also called playout or rollout.



4) Backpropagation: use the result of the playout to update information in the nodes on the pat. The simulation result is “backed up” (i.e. backpropagated) through the selected nodes to update their statistics.



These may be grouped into two distinct policies:

1) Tree Policy: Select or create a leaf node from the nodes already contained within the search tree (selection and expansion).

2) Default Policy: Play out the domain from a given non-terminal state to produce a value estimate (simulation).

Upper Confidence Bounds (UCB)

The algorithm UCB1 is an algorithm for the multi-armed bandit that achieves regret that grows only logarithmically with the number of actions taken. It is also dead-simple to implement, so good for constrained devices. [1]

$$UCB1 = \frac{Q_j}{N_j} + C\sqrt{\frac{\ln(N)}{N_j}}$$

Upper Confidence Bound applied to trees (UCT)

Kocsis and Szepervari (2006) first formalised a complete MCTS algorithm by extending UCB to minimax tree search and named it the Upper Confidence Bounds for Trees (UCT) method. This is the algorithm used in the mount of current MCTS implementations. [2] [5]

UCT may be described as a special case of MCTS, that is: UCT = MCTS + UCB.

4.Methodology

Collect data of state

In the provide code, I choice the OXO as my sample to collect data. As OXO is a two-player game, the code has already implemented two UCTs as two players to play this game. The only one different between them is the number of iterations, one is 100 and the other is 1000. We could collect the data of state of board, which player just move and who win the game finally.

Dataset

In the dataset, it should include the state of board with the sequence of player's move and the game result.

Move_1	Move_2	Move_3	Move_4	Move_5	Move_6	Move_7	Move_8	Move_9	Result
0	4	2	1	7	3	5	8	6	0
2	0	6	4	8	7	1	5	3	0
3	4	0	6	2	1	7	5	8	0
8	4	2	1	7	5	3	6	0	0
4	0	1	7	3	5	6	2	8	0
0	4	3	6	2	1	7	5	8	0
4	0	6	2	1	7	5	3	8	0

6	0	2	4	8	7	1	5	3	0
4	0	2	6	3	5	1	7	8	0
4	0	6	2	1	7	3	5	8	0
6	0	4	2	1	7	3	5	8	0
4	0	2	6	3	5	7	1	8	0
4	0	2	6	3	5	7	1	8	0
0	4	6	3	5	7	1	2	8	0
4	0	6	2	1	7	5	3	8	0
0	4	1	2	6	3	5	7	8	0
0	4	2	1	5	0	0	0	0	2
...	...	...	...	...	...	...	...	...	...

Figure 1: Dataset Structure

The procedure of generating the dataset:

In the program, we simulate the player_1 and player_2 by two UCT. And player_2 is always

```
if state.playerJustMoved == 1:
    m = UCT(rootstate = state, itermax = 1000, verbose = False) # play with values for itermax and verbose = True
    #m = UCT(rootstate = state, itermax = 1000, verbose = True)
else:
    m = UCT(rootstate = state, itermax = 100, verbose = False)
    #m = UCT(rootstate = state, itermax = 100, verbose = True)
```

first to move. Hence, I declare an array that is named “datafile_2” and initiate it. The first 9 elements represent the states of each move, the other one indicates the game result. In each move, I record the state to each column by sequence.

```
datafile_2 = [0,0,0,0,0,0,0,0,0,0]
index = 0
```

```
# data : record the state
if state.GetResult(state.playerJustMoved) == 1.0:
    #datafile_1.append([state.playerJustMoved, m, state.playerJustMoved])
    datafile_2[-1] = state.playerJustMoved
    break
else:
    #datafile_1.append([state.playerJustMoved, m, 0])
    datafile_2[index] = m
    index += 1
```

Furthermore, in game of OXO, when one of the players moves, I check the result of this game. if the player who just move have win the game, I will stop simulating the game and record the result. Finally, I use the function of to_csv of pandas to save the state of game in the .csv file named data_OXO_2.csv. And if there is another game state generating, I will record it by appending to the same file.

```
## save data as .csv file
if not os.path.isfile('data_OXO_2.csv'): # if file does not exist write header
    head_csv = True
else:
    head_csv = False

df_2 = pd.DataFrame([datafile_2], columns = ["Move_1","Move_2","Move_3","Move_4","Move_5",
            "Move_6","Move_7","Move_8","Move_9","Result"])
df_2.to_csv('data_OXO_2.csv',  mode='a', index = False, header = head_csv)
```

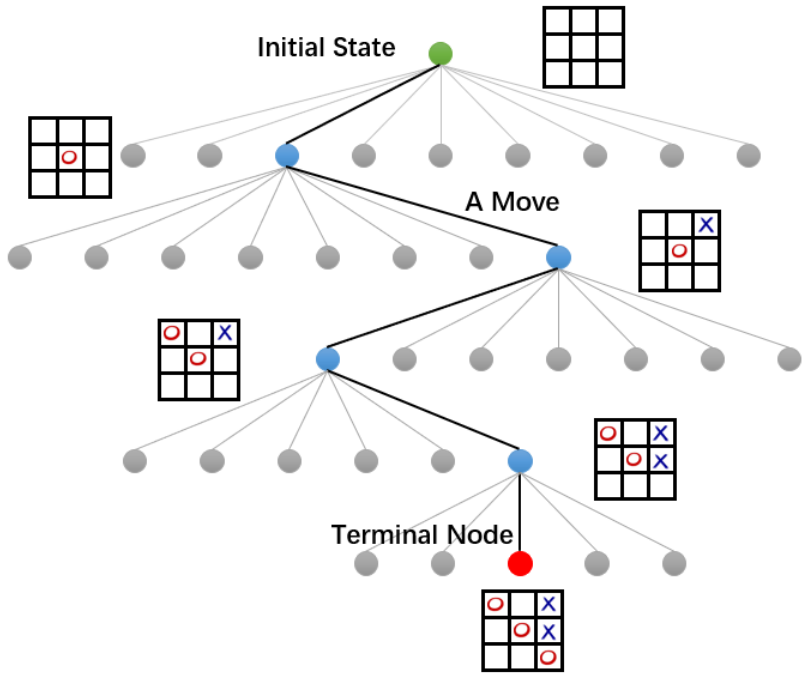
After this, I just need to make a loop that has 100 cycles for collecting data of game statement.

```
for i in range(0, 100):
    UCTPlayGame()
```

5.Experiments

A game tree is a tree in which every node represents certain state of the game. Transition from a node to one of its children (if they exist) is a move. The number of node's children is called a branching factor. Root node of the tree represents the initial state of the game. We also distinguish terminal nodes of the game tree – nodes with no children, from where game cannot be continued anymore. The terminal node's states can be evaluated – this is where game result is concluded. [4]

Describing the OXO game tree, you (partially) see:



- At the very top, you can see the root of the tree, representing the initial state of the OXO game, empty board (marked green);
- Any transition from one node to another represents a move;
- Branching factor of OXO varies – it depends on tree depth;
- Game ends in a terminal node (marked red);
- The tree traversal from the root to the terminal node represents a single game playout.

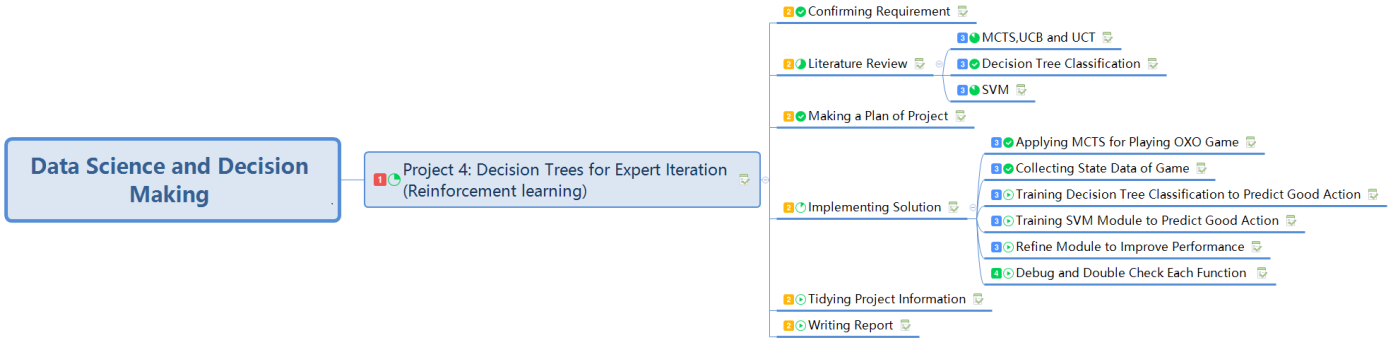
6.Conclusion

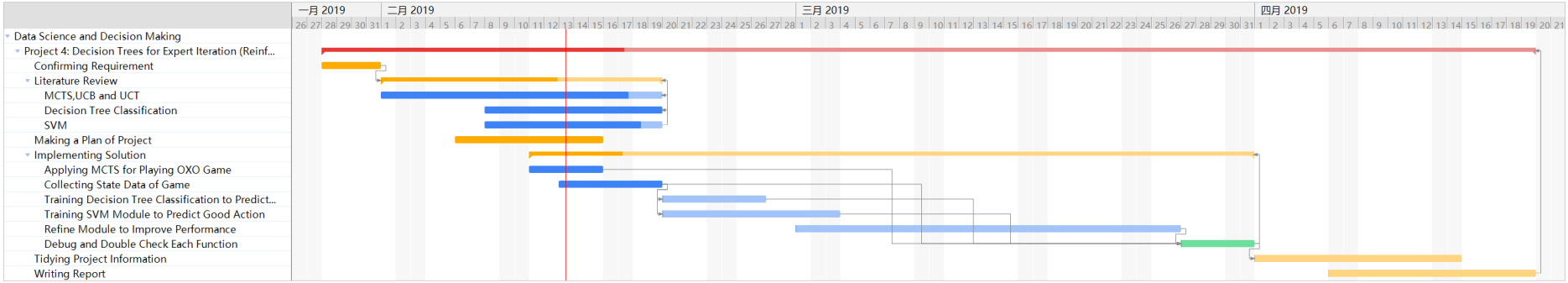
In task 1, I have studied the MCTS, and base on the provide code create a function to generate state of the game.

Then I can apply the data to train a module that can learn a fast classifier that is able to predict which action to take by machine learning approach. After that I will improve present program of MCTS. Refine the following approach to learn from the experience and make a better move instead of random choice.

Following task 1, I will train two kinds of machine learning approaches for comparing which has a better performance for predicting the move. Then apply the module to improve the MCTS. Because in each game, the MCTS module did not learn anything. It just search a better move based on current situation without any experience in the past games. However, if we replace the random rollout with a machine learning classification policy, the algorithm would have a better exploration probably. I will try to explore this part in following tasks.

7.Plan


Data Science and Decision Making
Project 4: Decision Trees for Expert Iteration (Reinforcement learning)
Confirming Requirement
Literature Review
MCTS,UCB and UCT
Decision Tree Classification
SVM
Making a Plan of Project
Implementing Solution
Applying MCTS for Playing OXO Game
Collecting State Data of Game
Training Decision Tree Classification to Predict Good Action
Training SVM Module to Predict Good Action
Refine Module to Improve Performance
Debug and Double Check Each Function
Tidying Project Information
Writing Report


一月 2019
二月 2019
三月 2019
四月 2019
Data Science and Decision Making
Project 4: Decision Trees for Expert Iteration (Reinf...
Confirming Requirement
Literature Review
MCTS,UCB and UCT
Decision Tree Classification
SVM
Making a Plan of Project
Implementing Solution
Applying MCTS for Playing OXO Game
Collecting State Data of Game
Training Decision Tree Classification to Predict...
Training SVM Module to Predict Good Action
Refine Module to Improve Performance
Debug and Double Check Each Function
Tidying Project Information
Writing Report

8.Provide references

[1]. Cameron Browne, Edward Powley, Daniel Whitehouse, Simon Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis and Simon Colton, “A Survey of Monte Carlo Tree Search Methods”, *IEEE transactions on computational intelligence and ai in games*, vol. 4, no. 1, march 2012

[2]. Jeff Bradberry, “Introduction to Monte Carlo Tree Search”, Mon 07 September 2015, online: https://jeffbradberry.com/posts/2015/09/intro-to-monte-carlo-tree-search/

[3]. Int8 about machine learning, “Monte Carlo Tree Search – beginners guide”, MAR 24, 2018, online: https://int8.io/monte-carlo-tree-search-beginners-guide/#UCT_in_Alpha_Go_and_Alpha_Zero

[4]. Anthony, Thomas, Zheng Tian, and David Barber. “Thinking fast and slow with deep learning and tree search.” In Advances in Neural Information Processing Systems, pp. 5360-5370. 2017.

[5]. Silver, David, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert et al. “Mastering the game of Go without human knowledge.” Nature 550, no. 7676 (2017): 354.