max-n-performance

April 21, 2020

[1]:
```
import pandas as pd
import numpy as np
```

1 Performance of maxN for Different Search Depths

The execution time of the maxN algorithm largely depends on the given search depth (how many turns the algorithm is allowed to look into all possible futures). However, even for the same search depth execution time differs quite strongly. Therefore, it is the purpose of this script, to find out, which parameters cause the execution time to go up.

1.1 Prerequisite

This script uses randomly generated games to measure performance. These random games were geneated by the node-script in `src/data-generation/random-games.ts`. The result is stored to two files:

- `data/random-games.json` which contains the game states for all games and moves (thus the complete game courses)
- `data/random-games.csv` which contains some analytical data to every move made in the random games (like number of players, number of pawns etc.)

Let's have a look at the analytical data:

[2]:
```
games_data = pd.read_csv('../data/random-games.csv')
games_data.head(3)
```

[2]:

	turn	playerOnTurn	boardSize	numOfPlayers	numOfPawns	numOfPawnsRed	numOfPawnsGreen	numOfPawnsYellow	numOfPawnsBlue	maxNumOfPawns
0	1	0	64	2	8	4	0	4	0	4
1	2	2	56	2	8	4	0	4	0	4
2	3	0	48	2	8	4	0	4	0	4

```
   numOfMovesCurrent  numOfMovesRed  numOfMovesGreen  numOfMovesYellow  \
0                 13             13                0                13
1                 13             26                0                13
2                 24             24                0                20

   numOfMovesBlue  numOfMovesAll
0               0             26
1               0             39
2               0             44
```

1.2 Measuring Performance

To measure the performance for each turn, another script was used. The script can be found at `src/data-generation/max-n-performance.ts`. It measured the execution time in milliseconds for every move from the random games. It did so with a search depth of 1, 2, 3 and 4 (4 already took a very long time to execute).

The results were stored to `data/max-n-performance.csv`. Let's have look:

[3]:
```
games_perf = pd.read_csv('../data/max-n-performance.csv')
games_perf
```

[3]:
```
     depth1  depth2  depth3  depth4
0         2       5      45     462
1         2       7      73     642
2         2      11     119    1151
3         2      12     115    1065
4         1      10     138     977
..      ...     ...     ...     ...
964       0       0       0       0
965       0       0       0       0
966       0       0       0       0
967       0       0       0       0
968       0       0       0       0

[969 rows x 4 columns]
```

1.3 Finding the Attribute with the Highest Correlation

Now, we try to find out if any one of the analytics (number of players, number of pawns, etc.) has a high effect on the execution time.

We will use a linear regression model with $x =$ each of our attributes and $y =$ each of the different search depths. We calculate the coefficient of determination (R^2) between an attribute and the execution time.

Let's see if some attributes have a stronger correlation to the execution time than others.

[4]:
```
def calcRsquared(x, y):
    corr_matrix = np.corrcoef(x, y)
    return corr_matrix[0,1] ** 2

def calcRsquaredAsDf():
    index = []
    columns = []

    table = []

    for keyData, valueData in games_data.iteritems():
        index.append(keyData)
        table.append([])
        i = len(table) - 1
        for keyPerf, valuePerf in games_perf.iteritems():
            rSquared = calcRsquared(valueData, valuePerf)
            table[i].append(rSquared)

    for keyPerf, valuePerf in games_perf.iteritems():
        columns.append(keyPerf)

    return pd.DataFrame(table,index,columns)

calcRsquaredAsDf()#.style.background_gradient(axis=0,vmin=0.2)
```

[4]:

	depth1	depth2	depth3	depth4
turn	0.163317	0.198774	0.141321	0.114319
playerOnTurn	0.000173	0.003025	0.000298	0.000348
boardSize	0.221690	0.443994	0.363315	0.301636
numOfPlayers	0.010049	0.108387	0.118162	0.107453
numOfPawns	0.155550	0.422254	0.388285	0.348615
numOfPawnsRed	0.096879	0.149143	0.115146	0.087310
numOfPawnsGreen	0.022468	0.179579	0.180872	0.173187
numOfPawnsYellow	0.159794	0.205633	0.179805	0.162279
numOfPawnsBlue	0.026578	0.197179	0.203254	0.193378
maxNumOfPawns	0.211422	0.306946	0.233411	0.189246
numOfMovesCurrent	0.409679	0.573768	0.464447	0.370929
numOfMovesRed	0.108454	0.191018	0.151513	0.118814
numOfMovesGreen	0.060835	0.334104	0.345835	0.338619
numOfMovesYellow	0.171485	0.291014	0.269541	0.246591
numOfMovesBlue	0.061444	0.350164	0.371416	0.348389
numOfMovesAll	0.215215	0.665986	0.643575	0.589013

1.4 Intermediate Results

The attribute with the highest correlation is the number of moves. This is quite logical as the number of moves is also equivalant to the branching factor.

Some other factors have an effect as well (number of pawns and board size). However, if we think about it, they support our hypothesis even more. The more pawns there are and the bigger the board is, the more moves can be made by the pawns. So, the number of moves remains the most important factor.

Different ways of counting the number of moves were used. There is the number of moves of the current player (who is currently on turn), the number of moves for each of the players (regardless whether they are on turn or not) and the sum of all possible moves of players.

For a search depth of 1, the number of moves of the current player is most significant (which makes sense because a depth of 1 means, we are only considering the currently possible moves). As soon as we start looking further into the future, the sum of all possible moves of all players is the most significant factor. This also makes sense as we have to compute all possible moves of the respective player for future moves as well.

***Conclusion*: We should determine the allowed search depth for the maxN algorithm by the number of moves *all* pawns on the board can make, regardless if the corresponding player is currently on turn or not.**

We continue in the next document: max-n-depth.pdf