Comparing different approaches of solving the 2048 game

Sanket Nawle (skn288@nyu.edu)

Manjiri Acharekar (msa530@nyu.edu)

Abstract

Our goal in this project is to use different approaches of solving the [1] 2048 game and to analyse how these algorithms perform. In our project, we compare modified versions of Depth First Search & Stochastic Hill Climber, and a new algorithm we developed ourselves.

Problem Description

[2] The game is comprised of a 4x4 grid, where each cell in the grid can hold a single block and in each block, appears a number which is an exponentiation of two. The game starts with two blocks on the grid with the values 2 or 4. Each turn the player can decide to move all the tiles on the board to one of four directions (up, down, left or right) only if moving the tiles causes the board to change. In each move, moving two tiles with the same number towards each other causes them to convert into a new tile whose value is the sum of the two previous tiles (this only happens if the two tiles have the same number). After each move a new tile is added in a random location on the grid (in one of the unoccupied cells with equal probability); this new tile has the value 2 with a probability of 0.9 and 4 with a probability of 0.1. The goal of the game is to keep connecting tiles until reaching the value 2048. You have successfully completed the game if the tile of 2048 is reached. The game ends if there are no more available moves, i.e., the player cannot make any move that changes the board and thus is stuck in the current state.

The 2048 game is a combination of 2 previously released and popular games Candy Crush Saga [3] and PushPush [4]

We have used the [5] 2048 game implementation created by Sondre Dyvik and Paul Philip Mitchell

Algorithms

Depth First Search (modified):

The algorithm is modified in the sense that it doesn't take the first move available directly. It sends all the available moves for the current state to a function. Now, this function works recursively and takes the first available move just like traditional DFS [6]. The main (run) function now calculates which move would be better by comparing the scores (heuristic) of each move

obtained from the recursive function and then it takes the best move accordingly. Thus, at every instance, each available move generates its own DFS tree and we compare the root of these trees to take the best move.

The pseudo code of the algorithm is as follows:

Run function:

For every next move

{

Make a copy of the board and analyse the available directions at that state.

For all the available directions

Compute the score by calling the function dfsRecursive and set the depth as zero initially

Take the move with the highest score

}

dfsRecursive function

Make a copy of the board obtained from the run function

Take the move

For all the possible moves available now

{

If there are no possible moves available or depth has reached the depth limit

Calculate the score of the current state and if it is better, return that score

Else go deeper in the search tree by sending that move to itself and increasing the depth

}

Stochastic Hill Climber (modified):

It is very similar to the Depth First Search strategy [7]. But here, in the recursive function we take any random move and if there is still some move possible, we go deeper in the search tree by taking that move. This algorithm performs the worst amongst all because of its randomness of the move taken. What multiplies its degrading performance is the randomness of the tile generated.

In the above two algorithms, we have not completely forwarded the game state. We have not generated a random tile after taking a move because the framework we are using is time independent. Thus, it is of no use to generate a random tile and go deeper in the search tree by assuming that the random tile will be generated at the same place. It is very likely that after taking a move, the search tree which was generated before using the recursive function is completely different than the one we would have now. Thus, we are only focusing on the tiles which can be merged to increase our score. Thus, having a large depth limit may not be beneficial since we are assuming that the existing tiles can get merged regardless of the random tile generated after each move. So, after some analysis, we have concluded that a depth of 5 works the best for DFS and depth of 10 works best for the Stochastic Hill Climber.

A new algorithm:

After multiple attempts on solving this game, we came up with a generalised idea of how we could increase our score. The focus of this algorithm is to pile up the heavier tiles at the bottom and try to keep the largest tile at the lower right corner of the grid. The pseudo code of the algorithm is as follows:

If the bottom row is empty and taking the downward move can fill the void

{Take the downward move}

Else for the row which has all distinct values, check if taking the left and the down move consecutively merges any tile from the row above it

Else if for each row,

{If taking the right move merges any tiles, take the right move

If taking the down move merges any tiles, take the down move}

Else if there is a right move possible, take it

Else if there is a down move possible, take it

Else if there is a left move possible, take it, and take right immediately in the next step if it is possible

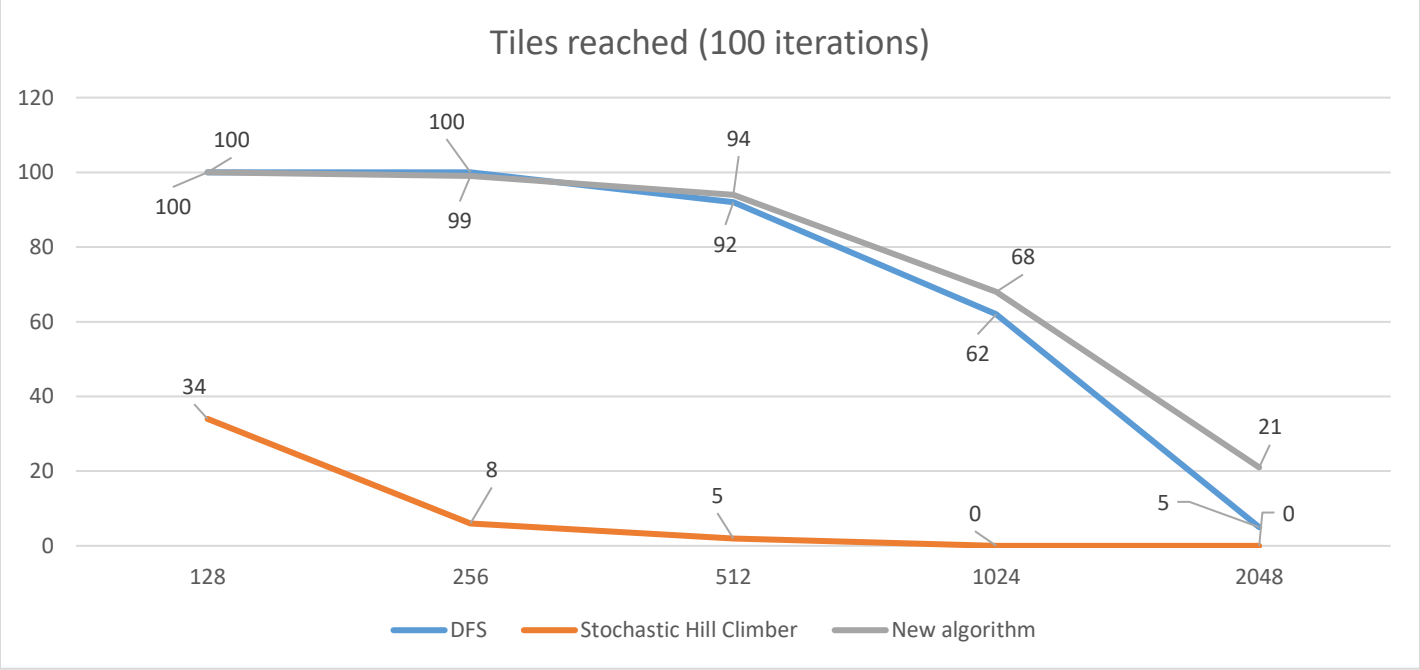
Else if there is an upward move possible, take it, and take down immediately in the next step if it is possible

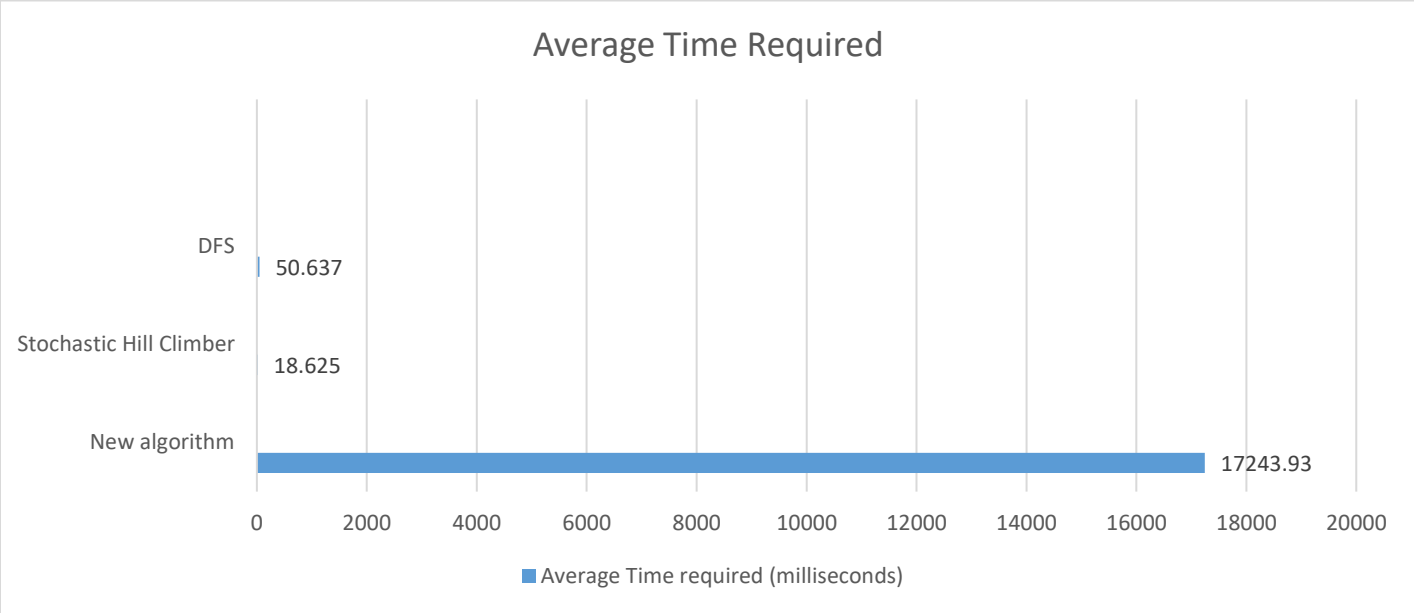
The above algorithm has a depth of zero, i.e., it takes the move based on the information of the current board state. The above algorithm plays better than DFS. The reason being the uncertainty

of the position of the random tile generated after progressing in the search tree in case of DFS. If that was not the case, DFS would have performed a lot better than the above algorithm.

Analysis:

The charts below represent the minimum, maximum and average scores each algorithm could achieve after 100 iterations and also the number of times the algorithms could reach 128, 256, 512, 1024 and 2048 tiles. Also, considering the average time required to complete one game, it can be seen that the average time required by DFS and Stochastic Hill Climber is negligible as compared to that of the new algorithm.





References:

[1] "2048," http://gabrielecirulli.github.io/2048

[2] http://www.cs.huji.ac.il/~ai/projects/2014/2048/files/Report.pdf

[3] Luciano Guala', Stefano Leucci, and Emanuele Natale. Bejeweled, Candy Crush and other Match-Three Games are (NP-)Hard. CoRR, abs/1403.5830, 2014.

[4] Erik D. Demaine, Martin L. Demaine, and Joseph O'Rourke. PushPush is NP-hard in 2D. CoRR, cs.CG/0001019, 2000.

[5] https://github.com/sondrehd/2048-ai

[6]http://s3.amazonaws.com/academia.edu.documents/35981597/dfs1971.pdf?AWSAccessKeyId=AKIAJ56TQJRTWSMTNPEA&Expires=1481690716&Signature=aAsB0tg35ToYA930KFJ5LVCwfoU%3D&response-content-disposition=inline%3B%20filename%3DDEPrH-FIRST_SEARCH_.ANI_LmEAR_GRAm_ALGOR.pdf

[7] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.19.3536&rep=rep1&type=pdf