The Core Algorithm of a Tetris-playing AI, and its Various Implementations

Matt Ritchie

CS360

5-9-18

Abstract

A Tetris-playing AI may seem like a simple program, what incorporates more than it may initially seem. At its core, it uses an algorithm designed to reduce the total volume of the objects it is assembling within a defined area. This concept can be applied to several different real world applications, not just Tetris. Furthermore, the core algorithm of a Tetris AI can be configured in multiple different ways, and does not follow only one procedure for all implementations.

Introduction

The usefulness behind investigating the algorithm within a Tetris playing program is that the algorithm is designed to compress space of whatever it is given, which in this case is Tetris pieces. You could then use this algorithm in anything that needed to minimize space, for example, if you developed it a little further so that it could play three-dimensional Tetris, you could use it to pack items in a box for shipment with the most efficient use of space. Furthermore, it could also be used in deciding how to stack irregularly shaped objects on shelves in a warehouse, such as oddly shaped engine parts in an auto repair store, or organize where to put specific parts in a hardware store. Such an algorithm may also have uses in computer chips as well, since RAM pulls in a block of memory, it would be most efficient if all of the needed data could be fit within that one block.

Implementation

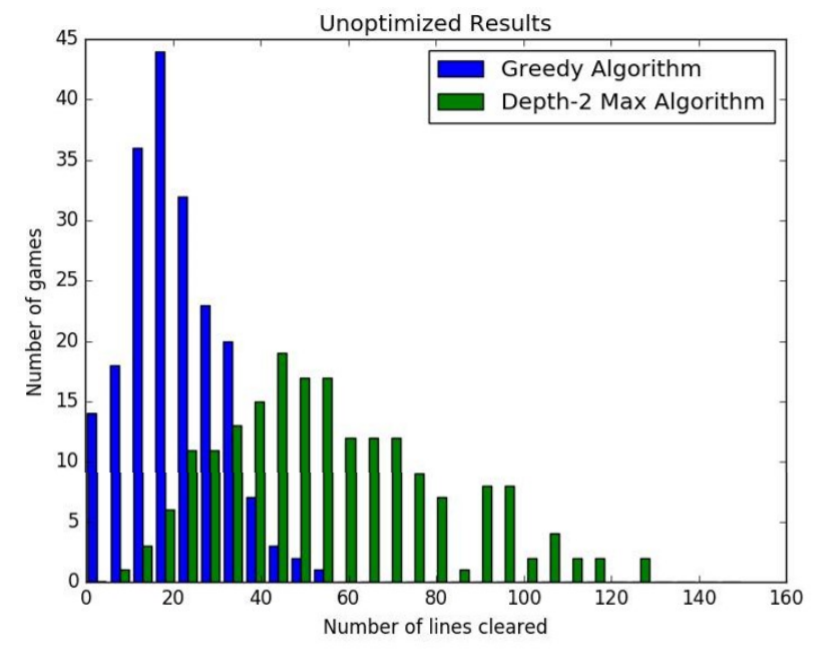
The basic process of the algorithm is to take into account four concepts, and then decide where to place the new Tetris piece while best adhering to those parameters. The first parameter is how many filled lines of squares the new piece in its potential position would create, the algorithm tries to maximize this value. The second parameter is the amount of holes in the remaining game bored, meaning open spaces that are not accessible from the surface, which the algorithm tries to minimize. Another parameter is the overall height of the surface of the game board, which is minimized by erasing filled rows. The last concept the algorithm takes into consideration is how uneven the accessible surface is, in that it tries to keep the surface as flat across the top as possible. These criteria are taken into consideration to calculate how good the potential position is for the current Tetris piece, and that value is what marks the best position for the new piece, out of all possible horizontal positions and rotations of the Tetris piece.

If the best position calculation is only performed on the current Tetris piece, the algorithm will run quickly, as it is an example of a greedy algorithm. A better implementation is to not only consider where the best place for the current piece is, but to also consider how well the current move sets up the game board for the next piece which is usually shown to the player one move in advance. This implementation runs much slower, because the algorithm must consider not only how the current peace fits in any potential position, but also how well the next piece would fit in any position afterwards. Taking subsequent Tetris pieces into account on any given current peace effectively makes the runtime of the decision-making algorithm exponentially larger for each additional piece available for preview.

Results

For the greedy type set up for the algorithm, the asymptotic runtime would be the number of possible rotations of the tetris piece, 4, times how many positions exist horizontally that it could be placed in, i.e. the number of columns on the game board. Given the shape of the specific Tetris piece there may be less potential horizontal positions than the number of columns, but the asymptotic upper bound could be found by the formula O(4c) where c is the number of columns, which would reduce to O(c).

The setup of the algorithm which is more like dynamic programming, involving the potential positions of the next piece, runs much slower because for each potential position of the current piece, the algorithm must consider every possible position of the next piece. This would produce in asymptotic behavior of O(4c\*4c), or O(c^2). Potentially it could be written as O(c^t) where t is the number of subsequent Tetris pieces available for preview, in the case that the AI or the player is capable of seeing more than one Tetris piece in advance.

Despite the difference in run times between the two described implementations, the greedy algorithm setup does not manage to clear as many lines as the implementation that considers the subsequent Tetris pieces as well, similar to the rod cutting problem we examined in class. The graph below is from a team from Stanford University exploring Tetris AI algorithms, and shows the actual test results of the two implementations (Kang 2).

Extensions

As far as improvements go, some developers have tried to incorporate genetic learning algorithms into their Tetris solving AI's. With enough time 4 the AI to learn what peace positions work best given certain conditions of the game board, the program could potentially remember what works best the last time a specific situation was encountered, and simply redo that solution, which would severely cut run times because the algorithm would not have to recalculate every possible position for every single move.

Conclusion

In conclusion, the current state of the art Tetris solving AI program is multi-threaded to allow the algorithm to play the game in real time like a player, instead of simply computing the best move before advancing the game. The core algorithm of this makes the most efficient use of a given space, which can apply to multiple uses in real life. It can be used to fit Tetris pieces together, pack boxes in the most efficient way for shipping, or potentially even reduce the disk space of programs on computer chips. While only taking one object at a time into consideration runs the quickest, the sacrifice of taking more time is entirely worth the results when the algorithm is allowed to take into consideration multiple objects, with the overall best result occurring when the program is able to view all objects in the set at the same time.

Works Cited

I. (n.d.). IdreesInc/TetNet. Retrieved from <https://github.com/IdreesInc/TetNet/blob/master/tetnet.js>

Kang, S. G., & Vo, V. (2017). AI Agents for Playing Tetris. Retrieved April 28, 2108, from <https://web.stanford.edu/class/cs221/2017/restricted/p-final/vtvo/final.pdf>

Tetris AI – The (Near) Perfect Bot. (2017, June 14). Retrieved from https://codemyroad.wordpress.com/2013/04/14/tetris-ai-the-near-perfect-player/