CS3243 Tetris Project

13-04-2017

1 Introduction

Tetris is likely one of the world's most famous and popular games. In this report, we describe how we devise an agent to play the game of Tetris. We use an agent that greedily picks the best possible next state from a given state, by using a heuristic function to approximate the value of a state. To train our heuristic function, we use a novel algorithm that is a combination of the well known Genetic Algorithm (GA) and Particle Swarm Algorithm (PSO). Our agent manages to clear 5 million lines on average with a max of 12 million lines, demonstrating that our algorithm is effective.

2 Agent Strategy

Our agent uses a linear weighted sum of features as the heuristic function for a given state. Given a state and a piece, the agent computes the heuristic value for all possible next states, and then greedily picks the next state with the maximum heuristic value.

//Insert Math equation

3 Features

We used the following features for our heuristic function:

- Altitude Difference: The difference between the height of the highest column and the height of the lowest column
- Number of Columns With Holes: The number of columns with holes, where a hole is defined as an empty square directly beneath a filled square
- Height of the highest column
- Number of holes in the entire board: Holes are defined as empty squares that have a filled square somewhere above it
- Number of Wells: The number of columns that have a height less than that of the 2 adjacent columns
- Rows cleared: The number of rows cleared for that particular move
- Total Column Height: The sum of the heights of all the columns
- Total Column Height Difference: The sum of the difference of heights between adjacent columns
- Column Transition: Number of transitions from filled squares to empty squares and from empty squares to filled squares within columns
- Deepest Well: Height of the lowest column

- Row Transition: Number of transitions from filled squares to empty squares and from empty squares to filled squares within rows
- Weighted Block: Sum of value of every square, where a square's value is equal to the row it is in (numbered from the bottom)
- Well Sum: Using the same definition of wells as above, this sums up the depth of every well

While running our training algorithms, we noticed that some features were more important than others. In particular, the algorithm assigned highly negative weights to Column Transition and Well Sum, while assigning highly positive weights to Number of Wells. This was slightly unexpected as we thought that Rows Cleared would have the largest positive weights, to predispose the algorithm towards clearing more rows. However, the weight for this heuristic varied wildly between positive and negative, indicating that perhaps sometimes the algorithm preferred to not greedily pick moves that were clearing rows, but rather chose to maintain an even surface at the top.

4 Our Algorithm

Our algorithm consists of a combination of genetic algorithm and particle swarm optimization (PSO) algorithm. Each algorithm is run as a separate island with a total population of a hundred set of heuristics with different weights. After every generation, a tenth of the population from each island will be migrated to the other island. A generation is defined as a sequence of crossover and mutation for the genetic algorithm, and as an update of the velocity and position of the PSO algorithm. We will first describe the working principle behind the genetic algorithm and the PSO algorithm. Subsequently, we will discuss the rationale behind why we chose to juxtapose these two different algorithms in our search for the best set of heuristics.

Genetic algorithm
Each generation of our genetic algorithm consists of the following sequence of steps

1. Evaluate fitness of each set of heuristics
2. Keep top half of population and cross-over the rest
3. Random mutation of each feature of crossed-over heuristics with probability of a tenth
4. Migration of a tenth of the population to PSO island

We keep the top half of the population to ensure that each set of heuristics that perform well will remain within the population. The mutation introduces some randomness to the genetic algorithm to avoid being trapped within a local maximum. The probability of mutation is set at ten percent to PSO algorithm

Rationale behind combination of algorithms

5 Experiments and Analysis

Diagram 1: Learning Diagram 2: Performance of agent
Experimentation 1. Architecture specifications on which the algorithm was implemented 2. The time taken to train 3. The performance of the agent
Analysis

6 Scaling to Big Data

We parallelised our algorithm by running PSO and GA islands on different cores, and futher by calculating the fitness of each population member on different cores. This is made possible as we ran our algorithm on the NSCC cluster, which contains 12 cores per compute node.

Parallelisation provides significant speedup to our algorithm. Previously, when we were running the algorithm on single thread, the time taken to compute the fitness values for the entire population of heuristic

can take up to 2 hours towards the later stage. Parallelisation allows us to distribute such time-consuming computation across multiple cores, effetively reducing the computation time by an order of magnitude.

To demonstrate the effect of parallelisation, we compared the speed performance of two versions of the algorithm, single-threaded and multi-threaded version, sharing same set of parameters. We started both algorithms as two separate jobs on the NSCC cluster at the same time, and allowed them to run for around 20 hours before comparing the results. The baseline for comparison is the total number of lines cleared in the entire time span (Explanation for choosing this??). The single-threaded algorithm clears a total of 158182993 lines, while the multi-threaded version clears a total of 1899941699 lines. The result shows that parallelisation provides about 12 times speedup, demonstrating that our we can achieve linear speedup by parallelising our program.

Talk about parallelising algorithm. Mention MPI Get speedup

7 Conclusion

8 References