Indian Institute of Technology, Bombay

Dual degree project

Barbicels: Towards a world-class Scrabble agent

Mihir Kulkarni

Department of Computer Science and Engineering

Supervised by

Prof. Shivaram Kalyanakrishnan
Department of Computer Science and Engineering

13 December 2016

Acknowledgments

I'd like to thank Eeshan Malhotra, one of the original authors of the interface and the agent of which we bootstrapped, who also worked with me and was a major contributor on this project.

I'd also like to thank Prof. Shivaram Kalyanakrishnan for all his guidance

Contents

1 Introduction **4**

2 Literature Review **5**

2.1 Architecture 5

2.2 Agents 5

3 Methods **7**

3.1 Platform 7

3.2 Control architecture 8

3.3 Features 9

3.4 Training 9

4 Results **11**

5 Future work **11**

5.1 Better features 11

5.2 Better value function 12

5.3 Optimizations 12

6 References **13**

Abstract

We are aiming to build a world-class Scrabble player. We design an agent for playing Scrabble that uses evolutionary techniques to learn a linear evaluation function. Currently, our agent achieves a 40% win rate against Quackle, a popular online Scrabble agent. We also build a client-server model for people to be able to deploy their own Scrabble agents, thereby contributing to the overall research community. We have found that a lot of seemingly minor details are key to implementing the player well, as compared to only implementing the machine-learning stage of the pipeline.

In this report, we also lay out plans to proceed in stage 2 of our project, namely, by incorporating techniques that use neural networks and reinforcement learning, with better features.

1 Introduction

Scrabble is a two-player game, where two players alternate on forming words on a 15x15 board by placing tiles from a rack, which contains 7 letters at the start of their turn. Each word earns a score equal to the weighted sum of the scores of all the letters it contains, where the weights are 1,2 or 3 depending on the square where the tile is placed. In addition, the score may also be multiplied by 2 or 3, depending on the squares the word covers. In addition, playing all 7 tiles at once (a "bingo") gives a player an extra fifty points.

Figure 1: Source: https://rootsoftheequation.files.wordpress.com/2014/04/help_scrabble.jpg

There are a total of 100 tiles, which are initially placed inside a bag. Before the game starts, each player draws seven tiles each from the bag. At the start of each turn, each player plays some subset of his tiles on the board. This must include at least one letter immediately adjacent to an already existing tile (unless it is the first move, in which case it must pass through the center square), and all new words formed on the board (words are maximal contiguous sequences of letters, read left to right or top to bottom) must belong to the Scrabble dictionary. The player with the maximum score at the end of the game wins.

Read the rules in detail at http://www.scrabblepages.com/scrabble/rules/

2 Literature Review

2.1 Architecture

The Scrabble agents we have seen follow a control architecture where they implement forward-simulation followed by simulation. This involves:

- Generating all possible moves at a given state
- Rating each move, usually by simulating (usually by Monte Carlo trials) possible alternating plays by the opponent and the agent
- Picking the top move to play

Note that there is no machine learning involved in these agents.

2.2 Agents

There are three major online Scrabble agents out there today.

- Maven
  This uses three types of algorithms to evaluate moves:
  - "Mid-game" (less than 9 tiles in bag): All moves are sorted by some value function, after which the best moves are evaluated using Monte-Carlo search (2-ply).
  - "Pre-endgame": this is similar to "mid-game", except that it also tries to yield a good end-game situation
  - "Endgame" (no tiles in bag): When there are no tiles in the bag, Scrabble becomes a game of perfect information. The agent uses the B* search algorithm to find the optimal strategy (against a perfect player)

- Quackle

  This uses an approach similar to the Maven agent's "mid-game" evaluation. The scoring function uses the move score, an estimate of the value of the tiles left on the rack and a 2-ply simulation. The current game score, number of tiles left, and simulation are used to calculate the win percent of each move

- ELISE

  ELISE uses an algorithm similar to Quackle. The key difference is that it's Monte-Carlo search can be deeper ($>=$ 3-ply). This also uses optimizations to remove impact of unstable states during simulations and prune the search tree. This is the current strongest agent we could find, achieving around 61% versus Quackle's championship player.

3 Methods

3.1 Platform

We inherited a playing interface from a previous course project, (link to report) which we have built upon. This had code for setting up the game, completely evaluating moves, interfacing between agents to play the game. However, it was not built for extensive testing, and had no separation of client and server.

We built a modular, object-oriented object interface that uses TCP connections to conduct games between agents. The platform functions in the following way:

- We start both agents and have them listen on pre-specified ports
- We run the interface program and tell it how many games to play. The program uses a config file for setting its various parameters
- The interface program initializes the game (board, racks and bags) and sends a message to the first player asking it what move it wants to play. This is sent as a JSON object via TCP. It also begins a timer for that player
- The agent reads the message, computes its move and sends it back to the interface.
- The interface checks the move's validity. This is done by seeing if the words formed by playing the tiles obeys Scrabble's rules and belong to the dictionary. Based on the move, the interface sends either
  - The new tiles to the current agent, if the move was an exchange move
  - The new game state, consisting of the board, bag and the next agent's tiles, to the next agent; if a valid move was indeed played
  - An error message to the current agent if an invalid move was played
- This process repeats until one player exhausts their tiles, or someone runs out of time. In this case, the interface sends a message reporting the results to both agents, and a new game begins.

The advantage here is that time saved, since thez setup (loading dictionaries and so on) needs to be carried out just the once, at the start of the entire series of games.

Since the design of the agent is completely independent from that of the agents, this also gives us a new test-bed to develop and test Scrabble agents on.

Typically, the average time taken to simulate one game between our agent and Quackle is of the order of 40 - 45 seconds, with certain individual moves contributing significantly more than others. With four cores, this comes down to around 10 per game.

3.2 Control architecture

- The agent we design is memoryless, and decides the best move to play looking at only the current states of the board and the rack.
- The agent uses precomputed dictionaries to generate all possible moves depending on the board and rack. This is currently done using a data structure called a DAWG (Directed Acyclic Word Graph).
- The dictionary has around 270,000 words, and the average number of moves generated per turn averages at around 300, but can be as high as 1000+. Moves can be either through hook (perpendicular to an existing word) or parallel (just adjacent to existing word) plays.
- We compute the score for each word using a certain scoring function. As of now, we have been able to test a linear agent, that computes features corresponding to each move, and computes it's score based on a dot product with a weight vector. We play the top-scored move.
- It is interesting to note that several parts of the control architecture were tricky for us to get right and optimize for making the agent perform reasonably. Clearly, there is much more to making a Scrabble agent than simply the learning algorithm.

3.3 Features

We use the following features right now

- `Current score difference`: This, we think, helps the player decide what sort of strategy to use. For instance, we'd expect someone to play more aggresively if they are far behind, and vice versa
- `Move score`: A higher score should be favoured
- `Difference in consonants and vowels left on rack` : We conjecture that this feature being small will mean that better moves can be played after drawing tiles
- `Number of blanks left on rack`: We think this will allow better moves to be played in the future.
- `Probability of drawing playable bingo next turn`: THis is computed using Monte Carlo trials right now. We iterate through a fixed number of trials, randomly drawing from the bag on each turn, and computing whether or not a playable bingo is drawn.
- `Expected playable bingos on next turn`: This is another feature (computed in the same way as the previous one) that helps us estimate how likely it is to draw a bingo

We are moving to an analytical computation of the bingo features, to save time spent on evaluating the Monte Carlo trials.

3.4 Training

We use an evolutionary approach, called CMA-ES (Covariance Matrix Adaptation Evolution Strategy), to find the best weight vector for our linear model. The idea of this method is to use the top few agents of a population to figure out the mean and covariance matrix of a Gaussian distribution, and use that distribution to generate the new generation of the algorithm. This proceeds as follows:

1. An initial weight vector $w_0 \in \mathbb{R}^n$ and an initial variance-covariance matrix $\Sigma \in \mathbb{R}^{nxn}$ are chosen. We use these values from the old, trained linear agent from the previous project we inherited.
2. A new Gaussian distribution $\mathcal{N}(w_0, \Sigma)$ is created
3. n weight vectors $\{w_i\}_{0<i\leq n}(n = 50$ for us) are drawn from this distribution
4. We evaluate each such agent (which computes score of a move with features ϕ as $w_i.\phi$) by playing m ($= 100$ for us) games
5. We score each agent by
    - Number of wins
    - Cumulative score difference (in case of ties in number of wins)
6. We pick the top 10 agents $\{w_i'\}$, and compute
    - Their mean w_m
    - Their variance-covariance matrix Σ_m
7. A new Gaussian distribution $\mathcal{N}(w_m, \Sigma_m)$ is created
8. If we have had sufficient iterations ("generations"), we exit, else, we go to step 2.

For our experiments, we ran 10 generations of CMA-Es with 50 agents playing 100 games each.

4 Results

We were able to achieve a best score of 40.2% against Quackle's fast player with our linear model, averaged over 500 games. We achieved the best agent after 5 generations, leading us to believe that CMA-ES will help us only upto this point, with the current feature set.

For comparision, a simple greedy agent (which plays the highest score each time) gives us around 29.5% against Quackle

5 Future work

5.1 Better features

As of now, we are using only five features in our model. (The first feature we mention is useless for a linear agent to arbitrate between moves). We have a list of features to further use in our agent. Some of these include:

1. `State-only features`, which are useless in arbitrating between moves in a linear model, but could help when different features interact with each other, say, in a neural network. For example,
    - `Tiles left in bag`
    - `Unseen consonants - unseen vowels`
    - `Unseen blanks`
2. `Number of tiles left after move`
3. `New Bingo lanes`
4. `New hotspots available`

5.2 Better value function

We plan to move from a simple linear model to a Neural Network to approximate the value function, since we believe that certain features do interact non-linearly to decide the value.

The idea is to train such a network either through either evolutionary techniques, or backpropogation, using deep reinforcement learning.

We wish to be able to learn the topology of the network by using something like the NEAT methodology. NEAT involves growing a neural network by starting with a simple one and gradually augmenting it with more neurons, using evolutionary methods.

5.3 Optimizations

- Perfect endgame play: The endgame has perfect information. Therefore, we will incorporate a deterministic endgame algorithm (which all our online agents do as well), to greatly improve endgame. This involves a minimax search over all possible game trajectories. While we have this ready right now, the move generation is too slow. This can be solved using better data structures, such as the GADDAG.
- GADDAG: Right now, we use a two-sided DAWG to generate the moves to play. This can be modified to a new data structure called the GADDAG, which speeds up move generation and will greatly help in speeding up the endgame tree computation.
- Analytical calculation of bingo probabilities: The Monte Carlo tree simulations we use currently to compute the features relating to bingo probabilities are very compute intensive. By replacing these with an analytical term (which assume a uniform distribution of yet-unseen tiles), we should be able to heavily cut down the time required for our moves

6 References

- ELISE Crossword Game Software, C. M. Street, *Scrabble program released in 2013.* (ELISE)
- World-championship-caliber Scrabble, Brian Sheppard, *Artificial Intelligence, 2002, Elsevier.* (Maven)
- Quackle, Jason Katz-Brown and John O'Laughlin, *Open source Scrabble program released in 2006.* (Quackle)
  Source: https://github.com/quackle/quackle
- The World's Fastest Scrabble Program, Andrew W. Appel and Guy J. Jacobson, *Commun. AC, 1988.* (DAWG)
- A Faster Scrabble Move Generation Algorithm, Steven A. Gordon, *Software Practice and Experience, 1994.* (GADDAG)
- The CMA Evolution Strategy: A Comparing Review, Nikolaus Hansen, *Towards a new evolutionary computation. Advances in estimation of distribution algorithms. pp. 75-102, Springer, 2006.* (CMA-ES)