Reproducing A Competitive Scrabble Agent

Casey Pore
Colorado State University
cpore@rams.colostate.edu

Problem Description

In this paper, I describe the process of building a Scrabble agent from scratch and evaluating its potential for competitive play. The unique properties of Scrabble make it difficult to apply existing generalized AI strategies common to other games, such as an adversarial search. In fact, even the best Scrabble agent (Brian Sheppard's Maven) does not utilize any local search techniques until the very end of the game, although it has been shown that the end-game search can be crucial for cinching a win against the best human players (Sheppard 2002). As we will see, creating a competitive Scrabble player is more about adding features that incrementally improve the agent's chances for higher scoring, such as a combination of move selection heuristics and end-game search, than it is any one over-arching concept of Artificial Intelligence. This sentiment was expressed by Brian Shepard when he said his program, Maven, "is a good example of the 'fundamental engineering' approach to intelligent behavior" (Sheppard 2002).

Scrabble is neither a game of perfect information nor a zero-sum game. Outcomes are highly dependent on the usefulness of the tiles a player randomly draws from the bag to make high scoring moves. Additionally, it cannot be known what tiles the opponent holds. This makes it difficult (or at least futile) for an agent to plan ahead or anticipate future states of the game from which to make informed decisions about how to play its tiles. This limits the options for quickly generating moves an agent may select to play. To accomplish the goal of generating possible moves quickly, an compact data structure along with an efficient algorithm for identifying valid moves lie at the heart of my implementation, as well as agents that have come before mine. Once this fundamental technique was in place, I worked toward adding heuristics to improve the chances of maximizing the average move score in hopes of drawing my agent closer to being as competitive a player as possible.

After completing the move generation algorithm, the first heuristic implemented was to simply choose the move which produced the the highest score. I found that my implementation, though slightly slower than Appel and Jacobson's player from which I based my implementation, outperformed their player in regards to average move score by a significant margin (Appel and Jacobson 1988). Next, I implemented several heuristics (taking inspiration from a few different sources) under the hypothesis that each heuristic would add a statistically significant increase to the average move score of my agent and increase its win ratio. An agent was created for each of these heuristics and pitted against an agent using only the basic maximum score heuristic to evaluate their effectiveness. Finally, after evaluating which of these individual heuristics led to a better outcome, I implemented several agents that use different combinations of the heuristics hypothesizing that they would lead to even better outcomes. These multi-heuristic agents were pitted against the maximum score agent to evaluate which combinations worked the best. I found that almost all of the multi-heuristic agents played better than any single-heuristic agent for creating strong Scrabble agent, however, not all of the single-heuristic agents bested the maximum score agent.

Previous Work

Techniques for solving the problems of creating strong AI Scrabble players has been investigated as far back as 1892. The earliest attempt that I encountered was a paper by Shapiro and Smith (S&S) called "A Scrabble Crossword Game Playing Program" (Shapiro and Smith 1982). This first attempt was limited by it's move generation algorithm. Moves were generated by trying permutations of tiles and then using backtracking to check the validity of the attempted move. Move positions were selected by placing words across existing words only. There was no intelligence about forming words with adjacent tiles to make multiple words, however the algorithm still produced moves that scored a respectable 13.125 points on average by selecting either the highest scoring move or the first move it found about some point threshold (Shapiro and Smith 1982). Despite its limitations, this paper laid the foundation for how future programs would achieve a fast search by representing the lexicon as a trie data structure combined with a backtracking algorithm to find valid words.

Though work to create Scrabble agents started over three decades ago, major advancements were sparse and far between. Scrabble as an AI topic is indeed a narrow niche and not popular among academics due to its clear lack of opportunities for improvement and general application. The next milestone in Scrabble agent improvement came in 1988 in a paper by Appel and Jacobson (A&J) that improved the speed over the fastest Scrabble agents at the time by two orders of magnitude (Appel and Jacobson 1988). A&J expanded upon S&S's program in two ways. First, the trie is reduced to a Directed Acyclic Word Graph (DAWG). While this does not directly improve the speed in which moves are generated, it has the advantage of significantly reducing the amount of memory needed to represent the lexicon. This made it small enough to keep the entire lexicon in memory, which did improve speed - an important enhancement considering the limitations of memory at the time, and still important today in mobile applications. Second, they implemented a better backtracking algorithm. S&S's program simply used backtracking to check if a permutation of letters was valid word. A&J's program improves this by considering information about the board to prune invalid moves from the search. This includes pre-computing cross-checks so that no tiles would be selected from the rack unless they were part of a valid cross-word, and using an ”anchor” to select where to start placing ”left parts” of a word to play[1]. An anchor is simply an open square with a tile to its right that a new word could be connected to.

A&J's program generates moves by using backtracking to place tiles from the rack that form left parts first. Because the cross-checks have been computed and there is a tile to build the left part from, there is never any tile placed from the rack that is not part of *some* word. These two properties reduce the number of moves that need to be placed and evaluated significantly. Once the left part of each move is formed, the algorithm builds the right part of the word in the same way, except accounts for tiles already on the board. The take-away here is that A&J's algorithm tries to maintain move validity as it goes. It ensures placed tiles are always part of a word, even if it can't complete it, whereas S&S's algorithm tries an entire permutation, and then checks that the move is valid. Like S&S's program, A&J's implementation simply selected the highest scoring move it found.

Finally, the apex in Scrabble agent development seems to have come in the form of Brian Sheppard's Maven. Development of Maven actually started in 1986 before A&J's program was created and was already a strong agent capable of tournament-level play (Schaeffer 2001). However, it immediately switched to using A&J's superior move generation algorithm when Sheppard discovered it. Maven is the accumulation of Sheppard's long history of building a Scrabble agent, and his experience is evident in its multi-faceted approach to play. Beyond A&J's fast move generation algorithm, Maven adds several features, many of which are applicable only to corner-cases, but important for catching every last bit of opportunity for selecting best moves. Sheppard has classified these features into 3 categories: rack evaluation, board evaluation, and search.

Rack evaluation is a mechanism by which the tiles left on the rack (known as a leave) for a given move are evaluated based on their utility to produce higher scoring subsequent moves. Two main techniques are used by Maven for rack evaluation. First, Maven maintains a static list of tile combinations, each resolving to a ”learned parameter” which adjusts the utility of the move based on the leave. The learned parameters are not a feature per se, but is the process by which Sheppard gathered statistics about what values are optimal to use with different combinations for evaluation. This parameter learning is not part of Maven (it does not employ any machine learning or adjustment of parameters at runtime), but were gathered as the result of ”a day's worth of self-play games,” tested for their optimality, and applied to the various leave combinations (Sheppard 2002). Beyond the learned parameters, Maven uses several heuristics (referred to as ”extensions” by Sheppard) for rack evaluation. These heuristics include potential usefulness of tiles left in the bag, vowel/consonant balance, and holding onto a ”U” tile if the ”Q” tile hasn't been played yet.

Board evaluation is the process of using features of the game board to determine the utility of a move. Several board evaluation techniques were considered. For example, it was hypothesized that choosing a good opening move that would prevent the opponent from playing on bonus squares. Or that you should always play a word on a bonus square, so that your opponent couldn't take it, even if it meant sacrificing a higher scoring move. In the end, it was found that almost all of the board evaluation techniques examined were irrelevant, except for one - triple word squares. Like the learned parameters for rack evaluation, Maven has a table of parameters to evaluate how many points could be compromised as a function of the placement of an opening move in terms of its location on the board.

Lastly, Maven performs a search during the pre-endgame and endgame. Before the endgame search, Maven performs a pre-endgame search which begins when there are nine tiles left in the bag plus the seven left on the opponents rack. Sheppard Explains very little about how the pre-endgame works other than it ”is a one-ply search with a static evaluation function that is computed by an oracle,” and admits that it may not be useful, except to prevent an opponent from fishing for bingos (using all seven tiles in one move). The endgame search commences when all the tiles have been drawn from the bag, so the remaining tiles on the opponent's rack can be deduced from what's on the board and the game becomes one of perfect information. The endgame search utilizes a B* algorithm that evaluates N-turn sequences of moves and returns intervals that correspond to the risk associated with the sequence, rather than the board state itself (Sheppard 2002).

Approach Taken

what did you do. Describe your project: what AI techniques are used by it, why you picked these techniques, how was the project structured, who did what, what sort of data was supplied (example problems for learning systems, prior models for non-learners), what results were expected. Code samples must be short.

-written from scratch

-except data structure

[1]Please note that for brevity, I only explain horizontal move generation throughout this paper. Vertical move generation is simply a transpose of these concepts.

-speed somewhere between SS and AJ
-strength somewhere between AJ an Maven

Move Generation

-Data structure
-algorithm

Heuristics

-MaxScore
-SaveCommon
-TileTurnover
-UseBonusSquares
-UseQ
-UWithQUnseen
-VowelConsonant

Evaluation and Analysis

How well did your project do: as expected, better or worse. Why did it perform as it did? What worked and what did not? Were there any surprises? What experiments/evaluation did you run? Include your experiment design to this section; it may have a subsection for the experiment design and another for the analysis.

Experiment Design

-define what makes a good player - max score is already very good
-

Experiment Analysis

Future Work, Conclusions

-performance enahncements
-android application
What did you learn? What would you do in addition or differently?

References

Appel, A. W., and Jacobson, G. J. 1988. The world's fastest scrabble program. *Commun. ACM* 31(5):572–578.

Schaeffer, J. 2001. A gamut of games. *AI Magazine* 22(3):29.

Shapiro, S. C., and Smith, H. R. 1982. A scrabble crossword game playing program. Technical Report 119, Indiana University.

Sheppard, B. 2002. World-championship-caliber scrabble. *Artificial Intelligence* 134(1):241–275.