An Artificial Intelligence that plays for competitive Scrabble

Fulvio Di Maria, Alberto Strade
Alma Mater Studiorum, Bologna
Master Degree in Computer Engineering
fulvio.dimaria@hotmail.it, alberto.strade@live.it

Abstract

An effective program created to play the worldwide famous crossword game SCRABBLE. It uses an efficient data structure that allows fast moves searching possible words trough the lexicon, while an heuristic function determinates the best word to be played, by means of probabilities and sensible considerations. Here is considered the two players version of the game.

1 Introduction

Scrabble is a famous board game, which requires a good knowledge of the native language and a good sense of strategy. On the contrary of other games like Chess and Go, Scrabble is an incomplete information game, since the opponent's rack of tiles is secret.

One of the first computer programs that played Scrabble was MONTY (1), which used both strategic and tactical concepts, but was also a bit slow and never managed to beat Scrabble experts. Others attempt were done, some using a straightforward strategy (playing the longest word or the one which gives more points), others approaching with a bit of protective strategy, like trying not to make available bonus squares to the opponent. In 1988 Appel and Jacobson managed, using a DWAG data structure (Directed Acyclic Word Graph) (2), to create the fastest and most efficient program of their time, surpassed later only by a variant called GALLAD.

While finding all the legal moves is a non-trivial problem on itself, because it implies a CSP (Constraint Satisfaction Problem),(3) being able to choose the best move to perform is equally difficult. Bayes' probability theorem allows to create a good heuristic function (4), choosing the best word evaluating also rack residues.

2 Scrabble Basis

The goal of the game is to make more points than the opponent. The board is composed of a 15X15 square rack where words has to be placed, like in crossword puzzles. There are 120 tiles representing the letters, and each of them reports the value of that letter; there also are 2 blank tiles, which can be used as any letter of the alphabet. Once they are put on the board, they become a copy of the chosen tile and cant be replaced, nor they have a value when calculating the score. Each word must cross at least one letter that was already on the board, and all the perpendicular words that come out from these crosses must be legal words. There are four types of bonus square: 3W triplicates the score of the entire word, 2W doubles it, 3L triplicate the value of the specific letter, while 2L doubles that value. When a player put his word, he draws from the bag until his rack returnes to 7 tiles. In the program at the moment we use a 8 tiles rack, giving human and cpu more possibilities to form longer words and making the game more interesting. Although it implies more computational load, the algorithm is quite efficient, so that this change does not dramatically affect the performance of the program. If a player manages to use all the tiles of the rack, he receives a 50 points bonus. After the tiles of the bag are finished the game turns in a complete information problem, but it ends only when one of the players runs out of tiles. Since we can divide each legal play in across and down, we can talk from now only about the across plays, because the board is symmetric.

3 Structures and Lexicon representation

Any across word must have some newly placed tiles and at least a tile already present on the board. Therefore, we can use this property in order to find legal words to add on the board. Lets call the leftmost newly covered square adjacent to a tile already on the board the *anchorsquare* for that word. So, all the possible anchors are the squares which are adjacent to the filled squares. This way we can reduce the problem of generating all the legal moves by dividing it for each row: for each of it, given the rack, the anchors and the tiles already placed, we can generate all the legal plays.

In order to perform a faster search, we need a structure that stores all the words of the given lexicon (which contains all the words of a dictionary, but also the declinations of the verbs, the plurals and so on). So we used a $trie$ (5), a tree which edges are marked by letters. If two words start with the same letters, then they share a portion of their path towards the ending node, which is marked as terminal. Note that all the leaves of the trie are terminal nodes, while the contrary is not always true. This structure can save a lot of words with a memory use proportional to O(n) bytes, resulting in a very efficient solution.

4 Possible words generation

The word generation can be easily divided in two phases: the detection of all the possible tiles anchored to the left of an anchor square (we call these tiles the $leftpart$ of the word); and for each of them, the search of all the possible *right parts* of the words, considering the right part all the tiles at the right of the anchor and the anchor square itself. Consider that the left part can be formed by tiles already on the board, or tiles from the rack, but not both. So, if the square adjacent on the left to the anchor is empty, we must put a left part from the rack (or even have a empty left part), then extend the word starting from the anchor square. If there are any tiles on the left of an anchor we can simply consider that as the left part and perform only the ”extend right” phase.

If the left part is all made of tiles already on the board, then we have already the left part, and we have only to note which is; otherwise, we have to find all the left parts. Since the anchor is the leftmost point of adjacency, the left part cant cover an anchor square, and that reduces the length of the left parts of a word anchored to a square, which will be found pruning the trie, according to the constraints of the tiles in our rack.

Once we have all the left parts, we can extend it on the right, adding the tiles one by one, according to the constraints, which are the residual tiles on the rack and the tiles already placed on the board to the right of the anchor square, which must be included in the word (if is not possible, we will immediately delete that istance of right part). Now we have all the possible words that can be played.

5 Heuristic

After the program has determined all the possible words that can be placed, an heuristic function has the duty to choose the most suitable word to play. In the program, the decision is based on 3 parameters, which influence the choice with different weights.

The formula used to express the fitness of each word can be expressed as $F = s - p + c$, where s is the score resulting from playing that word on the board; p is a non negative penalty score, which is greater if the move is weak from a defensive point of view; and c is a combo score, representing the possibility to put a 8 letter word on the table the next turn, which gives a 50 points bonus.

5.1 Penalty Score

Penalty score P is a score which defines how bad is the move from a defensive point of view, in other words it describes the bonus squares that the actual move allows the opponent to use. For each letter it places for the possible word, the program controls along the vertical axis if there are any bonus squares, starting from 7 squares above the word to 7 squares under the word. The malus is higher if it includes a triple score square, decreasing for a double word square, a triple letter square and a double letter square, but it is also conditioned by the distance of that square from the word (since it takes a longer word from the opponent to get advantage of the particular bonus square). Moreover, the malus is then multiplied by a number n, with $0 < n < 1$, which starts from 1 and decreases every time a letter from another word is encountered during the exploration (it takes a lot of effort to get a word long enough to reach the bonus square using a letter already on the board).

The penalty score then is the sum of all the maluses obtained from the exploration of the space up and down of each letter. This number can usually vary from 0 (perfect play from a defensive point of view) to 10 or more. The average value however, is about 6/7.

5.2 Bingo Probability

Before explaining the meaning of the c score, we must define an octet as a string which contains in lexicographic order the letters of a 8-word letter (eg. the word ABBAGLIO has as correspondent optet AABBGILO), and residual rack similarly. The value of c, given that in our lexicon there are over 10000 words with 8 letters, is expressed as

$$c = \sum_{i=0}^{10000} p_i * k$$

where p is the probability of having, after the draw, all the letters of the octet i, and k is a corrective value that is set to 6 after some experimental games. The value of p_i can result easier to understand with an example.

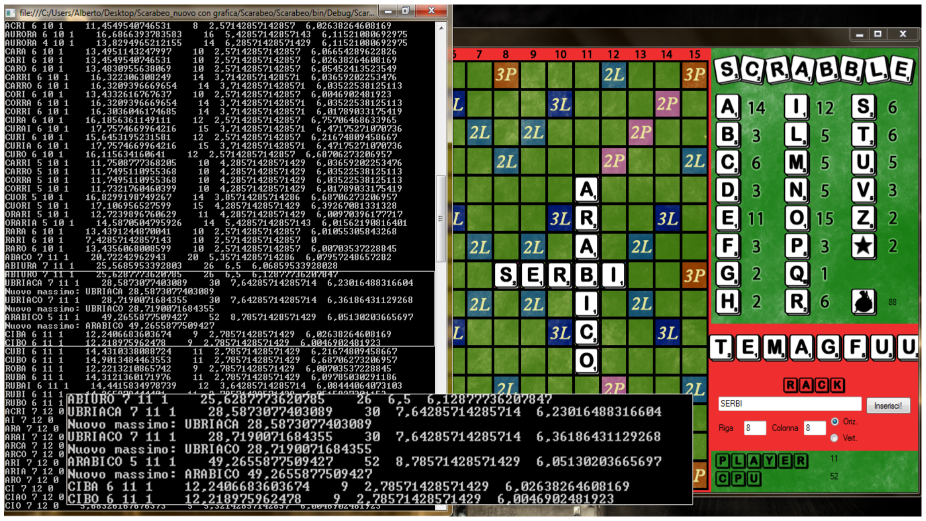
Given a rack, lexicographically ordinate, $\{$ABBEILMU$\}$, assuming that the program is evaluating the fitness of the word $\{$LUME$\}$ as the first word of the game, the residual rack will be $\{$ABBI$\}$. As for the word $\{$ABBAGLIO$\}$, the difference between its octet and the residual rack is $\{$AABBGILO$\}$ $\{$ABBI$\}$ = $\{$AGLO$\}$, so it needs to draw these 4 letters in order to have the complete word in hand. Lets consider now the remaining tiles pile, which is given by all the tiles in the game minus the ones that are on the board and in the residual rack. It will be something like this: $\{$AAAAA...GG..LLLL...OOOOOO...ZZ$\}$. Note that this is not the exact count of the remaining tiles, since we don't nknow which tiles the opponent has in his rack. Given the quantity available of each letter in the tiles pile, the probability of the desired draw is

$$\left[\prod_{i=A}^{Z} \binom{a_i}{d1_i}\right] / \binom{t}{d2}$$

where a is the quantity of tiles of that letter that stills available, $d1$ is the quantity of those tiles that have to be drawn, t is the total number of tiles and $d2$ the number of tiles to draw from the bag. In this case the p_i for the word ABBAGLIO will be $[\binom{9}{1} * \binom{2}{1} * \binom{4}{1} * \binom{12}{1}]/\binom{98}{4}$.

To enhance performances of c, since binomial is not a simple operation, it is possible to create a "binomial matrix" which stores all the binomial coefficients we will need during the game. It is also possible to check before the calculation if there are any chances for that word to be formed drawing from the bag. If there is not, the method skips to the next word and assign p_i=0.

Given this algorithm, is very easy to change it to make it fit with the original version of the Scrabble game, since we have just to change the octet with a septet in the program. The number of 7-letters words is higher than the number of 8-letters ones (15000 against 10000), but the chance to skip a good number of checks makes it possible not to degrade the performances of the program. Actually, the program takes between 3-7


file:///C:/Users/Alberto/Desktop/Scarabeo_nuovo con grafica/Scarabeo/Scarabeo/bin/Debug/Scar...
SCRABBLE
A 14 B 3 C 6 D 3 E 11 F 3 G 2 H 2
I 12 L 5 M 5 N 5 O 15 P 3 Q 1 R 6
S 6 T 6 U 5 V 3 Z 2 ★ 2 88
TEMAGFUU
RACK
SERBI
Inserisci!
Riga 8 Colonna 8
Oriz.
Vert.
PLAYER 11
CPU 52

```
ACRI 6 10 1      11,4549540746531     8  2,57142857142857  6,02638264608169
AURORA 6 10 1     16,6866393783583     16  5,42857142857143  6,11521080692975
AURORA 4 10 1     13,8294965212155     14  6,28571428571429  6,11521080692975
CARA 6 10 1      13,4951143247997     10  2,57142857142857  6,06654289622826
CARI 6 10 1      13,4549540746531     10  2,57142857142857  6,02638264608169
CARO 6 10 1      13,4830955638069     10  2,57142857142857  6,05452413523549
CARRI 6 10 1      16,322306308249     14  3,71428571428571  6,03659202253476
CARRO 6 10 1      16,3209396669654     14  3,71428571428571  6,03522538125113
CORI 6 10 1      13,4332616767637     10  2,57142857142857  6,0046902481923
CORRA 6 10 1      16,3209396669654     14  3,71428571428571  6,03522538125113
CORRI 6 10 1      16,3036046174685     14  3,71428571428571  6,01789033175419
CURA 6 10 1      16,1856361149111     12  2,57142857142857  6,75706468633965
CURAI 6 10 1      17,7574669964216     15  3,71428571428571  6,47175271070736
CURI 6 10 1      15,6453195231581     12  2,57142857142857  6,21674809458667
CURIA 6 10 1      17,7574669964216     15  3,71428571428571  6,47175271070736
CURO 6 10 1      16,115634160641    12  2,57142857142857  6,68706273206957
CARRI 5 10 1      11,7508777368205     10  4,28571428571429  6,03659202253476
CARRO 5 10 1      11,7495110955368     10  4,28571428571429  6,03522538125113
CORRA 5 10 1      11,7495110955368     10  4,28571428571429  6,03522538125113
CORRI 5 10 1      11,7321760460399     10  4,28571428571429  6,01789033175419
CUOR 5 10 1      16,8299198749267     14  3,85714285714286  6,68706273206957
CUORI 5 10 1      17,106956527599     15  4,28571428571429  6,39267081331328
ORARI 5 10 1      12,7239896760629     11  4,28571428571429  6,00970396177717
ORARIA 5 10 1     14,5870504795926     14  5,42857142857143  6,01562190816401
RARA 6 10 1      13,4391244870041     10  2,57142857142857  6,01055305843268
RARI 6 10 1      7,42857142857143     10  2,57142857142857  0
RARO 6 10 1      13,4356068008599     10  2,57142857142857  6,00703537228845
ABACO 7 11 1      20,7224296294 3    20  5,35714285714286  6,07957248657282
ABIURA 7 11 1     25,5685953392803     26  6,5  6,06859533928028
ABIURO 7 11 1     25,6287773620785     26  6,5  6,12877736207847
UBRIACA 7 11 1     28,5873077403089     30  7,64285714285714  6,23016488316604
Nuovo massimo: UBRIACA 28,5873077403089
UBRIACO 7 11 1     28,7190071684355     30  7,64285714285714  6,36186431129268
Nuovo massimo: UBRIACO 28,7190071684355
ARABICO 5 11 1     49,2655877509427     52  8,78571428571429  6,05130203665697
Nuovo massimo: ARABICO 49,2655877509427
CIBA 6 11 1      12,2406683603674     9  2,78571428571429  6,02638264608169
CIBO 6 11 1      12,218975962478     9  2,78571428571429  6,0046902481923
CUBI 6 11 1      14,4310338088724     11  2,78571428571429  6,21674809458667
CUBO 6 11 1      14,9013484463553     11  2,78571428571429  6,68706273206957
ROBA 6 11 1      12,2213210865742     9  2,78571428571429  6,00703537228845
RUBA 6 11 1      14,3121360171976     11  2,78571428571429  6,09785030291186
RUBAI 6 11 1      14,4415834978739     12  3,64285714285714  6,08444064073103
RUBI 6 11 1
RUBO 6 11 1
ACRI 7 12 0
AI 7 12 0
ARA 7 12 0
ARAI 7 12 0
ARCA 7 12 0
ARCO 7 12 0
ARI 7 12 0
ARIA 7 12 0
ARO 7 12 0
CI 7 12 0
CIAO 7 12 0
CIO 7 12 0
```

```
ABIURO 7 11 1      25,6287773620785     26   6,5   6,12877736207847
UBRIACA 7 11 1      28,5873077403089     30   7,64285714285714   6,23016488316604
Nuovo massimo: UBRIACA 28,5873077403089
UBRIACO 7 11 1      28,7190071684355     30   7,64285714285714   6,36186431129268
Nuovo massimo: UBRIACO 28,7190071684355
ARABICO 5 11 1      49,2655877509427     52   8,78571428571429   6,05130203665697
Nuovo massimo: ARABICO 49,2655877509427
CIBA 6 11 1      12,2406683603674     9   2,78571428571429   6,02638264608169
CIBO 6 11 1      12,218975962478     9   2,78571428571429   6,0046902481923
```

Figure 1: Our program in action. Thanks to its heuristic, it has found a word that has a high score (52 points) and does not leave high bonus squares to the opponent.

seconds to choose a move every turn, depending of the openings on the board and the quantity of possible words to test. If the program wouldn't use any heuristic it would take less than a second to find the best word to play (the one that gives the best score).

6 Improvements

Others strategies could be used to enhance quality of play of our program. It could evaluate the usege of large spots of the board, making it difficult for the opponent to find space where to place his word. It could change its strategy according to the score difference between its and the opponent's one. Since the "fitness" score is given as a sum of 3 values, giving each of them a different weight will create different styles of play. Prioritizing the score function, the AI will make more aggressive moves, trying to get as points as possible, leaving, however, good spots for the opponent; giving an higher value to the penalty score we will make the program produce more careful moves, trying not to give the opponent the chance to get a good score; emphasizing the octet function there will be more combo moves, maybe putting shorter words on the table in order to get a "bingo play". With the current set of values, the program can easily reach the 500 points threshold, offering a good challenge even for the most competitive players.

References

[1] Cosma, J.Jasckson: Introducing MONTY plays scrabble. Scrabble Player News, 7-10 (1983)

[2] Appel, Jacobson: The World's Fastest Scrabble Program. Communications of the ACM 31(3), 572-578 (1988)

[3] Russel, Norvig Ariticial Intelligence, a modern approch (3rd Edition). Pearson (2011)

[4] Richards, Amir:Opponent modeling in Scrabble.Proceeding of the Twentieth International Joint Conference on Artificial Intelligence, 1482-1487 (2007)

[5] Fredkin: Trie Memory. CACM 3.9, 490-500 (1960)