Scrabble on the TeraScale

BRIAN J WICKMAN
University of NebraskaLincoln

1. INTRODUCTION

1.1 Problems

Over the years, Scrabble r has become arguably the world's most popular word game. Unlike Chess or Go, Scrabble is a game of imperfect information as there are tiles left unseen throughout play. Therefore, adding search or strategy using standard techniques is infeasible.

Several authors in academia have produced programs [1, 2, 4, 5]. There are several limitations to these programs, however, most notably in their age. Consequently, there is little to no literature on harnessing the power of supercomputers for purposes of Scrabble play or research as the advent of highly available computing resources came at a later date.

There is also a lack of useful statistics for Scrabble players. James Cherry, author of an on-line Scrabble computer, simulated 5000 games and presented a body of statistics on it such as means, modes and standard deviations of winning game scores, aggregate game scores, highest scoring moves, number of bingoes per game, number of moves per game, and others. Statistics invaluable to wordgame programmers involve probabilities such as how often words are played. For example, the word QAT is played, on average, in 1 in 5 games using the Official Scrabble Players Dictionary. While a sample set of 4000 games can provide some clues into typical gameplay, samples of millions or hundreds of millions are required to achieve asymptotic results and to pan out any irregularities in the data.

1.3 Considerations

To liken Scrabble to a science, one needs to define precise rules of gameplay. First and foremost, a standard lexicon needs to be defined. In the United States, the Official Scrabble Players Dictionary or OSPD is used, whereas much of the world uses the Official Scrabble Words or OSW. The combination of these lists, coined SOWPODS, is used for World-Championship play. Secondly, tournament Scrabble is played one-on-one. Thirdly, each player has 25 minutes to make all his or her moves. The remaining set of rules differs on style of play. These rules include constructs for challenging words off the board, exchanging tiles, passing, and various other aspects important in computer play. It is assumed the reader knows the basic abovementioned rules.

2. MOVE GENERATION

2.1 Overview

The primary concern of a Scrabble programmer is to have a fast move generator, especially considering only a fraction of the total time consumed is not within this routine. Certain algorithms suffer in speed at the gain of compactness, while others do just the opposite. There are also complexity considerations. While a certain move generation algorithm may look good on paper, actual performance on an architecture depends heavily on the cache and branch-prediction mechanisms.

2.2 The Appel-Jacobsen Move Generation

1.2 Goal

The goals of the author are focused into one program, Martin Landau, which is a fully-functional Scrabble program both aggressively optimized and written for use in supercomputing environments. Its primary purpose thusfar is to generate sample games and consequently compute relevant statistics from these data. Results of such computations will be presented later in the text.

Algorithm

Appel and Jacobsen published a paper in 1988 that provided the first highly specific description of a Scrabble move generator. Their first insight was to store the lexicon in a trie and subsequently shrink it into a minimal graph via a finite state machine minimization algorithm. The resulting data structure is referred to as a Directed Acyclic Word Graph or DAWG. The minimization algorithm achieved nearly a four-fold reduction in size, allowing Appel and Jacobsen to fit the entire lexicon in core memory. Translating to today's machines, we can fit the same data structure almost entirely within cache.

The first step in the algorithm involves two precomputation steps. First, it finds the squares directly contiguous to laid tiles on the board. These squares are called anchor squares and are those on which we start placing tiles. The second precomputation step computes cross sets for each possible

	h_0	h_1	h_2	h_3	h_4	
v_0	B	O	I	N	G	v_1
	h_5	h_6	h_7	h_8	h_9	

h_0	= {A, O}
h_1	= {B, D, G, H, I, J, K, L, M, N, O, P, S, T, W, Y, Z}
h_2	= {A, B, D, G, H, L, M, O, P, Q, S, T, X}
h_3	= {A, E, I, O, U}
h_4	= {A, U}
h_5	= {A, E, I, O, Y }
h_6	= {B, D, E, F, H, I, M, N, O, P, R, S, U, W, X, Y }
h_7	= {D, F, N, O, S, T }
h_8	= {A, E, O, U, Y }
h_9	= {I, O, U}
v_0	= { }
v_1	= {S}

Figure 1: Cross Sets for BOING using SOWPODS

size of the lexicon data structure, this nondeterminism can be eliminated completely.

2.3 The Gordon Move Generation Algorithm

Steve Gordon's move generation algorithm is very similar to the Appel and Jacobsen algorithm. It uses the same minimal data structure but with a slightly different lexicon. For each word, it stores all its possible linear transpositions, separating wrapped letters with a delimiter. For example, APPEAR would be stored as six separate words: APPEAR*, PPEAR*A, PEAR*PA, EAR*PPA, AR*EPPA, and R*AEPPA. While this will incur a tenfold increase in the uncompressed size of the lexicon, the finite state minimization routine is able to to reduce the trie to 15% of its original size. A DAWG with the above transpositions is playfully called a DAGGAD.

To generate moves, it again starts at an anchor square but starts placing tiles to the right. If the tiles APPEAR are on the board, it will take the path APPEAR in the DAGGAD. The exit nodes from R are *AEIS. By taking *, it returns back to the anchor square and from then on places tiles backwards from right-to-left. For example, APPE AR*ER, a valid path in the DAGGAD, would append the prefix RE onto APPEAR to form REAPPEAR.

anchor square. If one is placing tiles on the board from left-to-right, he or she wants to lay tiles that also form valid vertical words. The cross sets on the board with the word BOING using SOWPODS are illustrated in Figure 1.

If the player had the rack AAEPPRS, he or she could bingo by placing, left-to-right starting at h_0, APPEARS. This is valid because A h_0, P h_1, P h_2, E h_3, and A h_4. Similarly, we could place, top-to-bottom starting at v_1, SARAPE because S v_1. v_0 is empty because BOING has no single-letter prefixes. Any tile may be placed on a square without a cross set so long as it is attached to a word intersecting an anchor square.

Once the above precomputation steps have occurred, Appel and Jacobsen's algorithm now considers the board one row at a time. By having the cross sets, this consideration is rendered entirely one-dimensional. Similarly, it transposes the board so that columns are considered in the same manner as rows.

From the position of each anchor square, it generates all possible left prefixes that can be formed from tiles on the rack, taking cross sets and paths in the graph into consideration. With the rack AAEPPRS, it may generate PRE, RE or PAE, but not PAER because, starting from the root of the DAWG, there is no path PAER.

For each of these prefixes, the algorithm tries placing the remaining tiles to the right, again considering cross sets and paths in the graph into consideration. Any time the last tile placed is a terminal node in the graph and the square directly right of the tile is blank, a legal move has been generated.

There is one big problem, however. The algorithm unnecessarily computes potentially hundreds or thousands of unusable prefixes. At the cost of an order of magnitude in the

The pseudocode for this algorithm is given in Figure 2 as a corresponding pair of recursive functions, same as that described by Gordon.

The author implements this algorithm nearly verbatim in Martin Landau.

Figure 3 and Figure 4 are reprinted from [3], with the author's permission, to illustrate the speed-size tradeoffs between basic implementations of both procedures.

2.4 Atomicity of Move Generation Algorithm

On the average, it takes Martin Landau approximately 0.0025 seconds to generate all possible moves given a random board and random rack on a single processing element of the Prairefire supercomputer. For racks with a single blank, time is increased six-fold. For racks with two blanks, time is increased an additional four-fold.

Because this number is so small, parallelization of the move generation algorithm on a system with relatively high-latency interconnect will actually decrease its speed considerably.

3. MATING SUPERCOMPUTERS AND SCRABBLE TO IMPROVE STRATEGY

A common misconception of novice Scrabble players results in them asking why one should implement a Scrabble program for use on supercomputers if existing programs already pick the highest scoring move at each turn. The primary problem with that impression is that there are strategic considerations missing. For example, playing AN for 3 might be better than SEXTANT for 93 if your opponent plays QUIXOTIC through your T for 365. Not only does the computer need to be aware of its offensive capabilities, but also it needs to be defensive as well.

```
Gen(pos, word, rack, arc)
// pos = offset from anchor square
{IF a letter L is already on this square
```

	DAWG per move	GADDAG per move	Ratio (D/G)
CPU time			
Expanded	1.344s	0.518s	2.60
Compressed	1.154s	0.489s	2.36
Arcs traversed	26, 134	10, 451	2.50

```
    GoOn(pos, L, word, rack, NextArc(arc, L), arc)
  ELSE IF letters remain on the rack
    FOR each letter, L, on rack and cross set
      GoOn(pos, L, word, rack - L,
           NextArc(arc, L), arc)
    IF the rack contains a blank
      FOR each letter, L, on cross set
        GoOn(pos, L, word, rack - BLANK,
             NextArc(arc, L), arc)
}

GoOn(pos, L, word, rack, NewArc, OldArc)
{IF pos <= 0 // moving left

  { word = L + word

    IF we are on a terminal, no letter directly left
      Record play
    IF NewArc != EMPTY
    { IF room to the left to place tiles

        Gen(pos - 1, word, rack, NewArc)
      NewArc = NextArc(NewArc, *) // shift directions
      IF NewArc != EMPTY, no letter to the left
        Gen(1, word, rack, NewArc)
    }
  }ELSE IF pos > 0 // moving right

  { word = word + L

    IF on a terminal, no letter directly right
      Record play
    IF NewArc != EMPTY and room to the right
      Gen(pos + 1, word, rack, NewArc)
  }
}
```

Figure 2: Pseudocode for the Gordon Move Generation Algorithm

	DAWG	GADDAG	Ratio (G/D)
States	17, 865	89, 031	4.99
Arcs	49, 341	244, 117	4.95
Expanded			
Bytes	1, 860, 656	9, 625, 648	5.17
Bits/char	29.5	152.6	
Compressed			
Bytes	272, 420	1, 342, 892	4.93
Bits/char	4.3	21.3	

Figure 3: Relative sizes of data structures

Anchors used	126.04	76.73	1.64

Figure 4: Average performance of DAWG and DAGGAD algorithms playing both sides of 1000 random games on a VAX4300

3.1 Monte Carlo simulation

Monte Carlo simulations are those which involve stochastic sampling to give good approximations of difficult to measure quantities. For example, to compute stochastically, one would generate N random points within the unit box and calculate the ratio of how many of these points fell within the unit circle. Benefits of these methods to parallel computing are their easy parallelizability and typically linear scalability.

To adapt Monte Carlo methods to Scrabble, we literally have to do some guessing. Given a particular board and rack, we may guess with varying degrees of success what our opponent's rack is. The more tiles on the board, the better chance of guessing their tiles. Suppose we make 1000 guesses for what rack our opponent has. Now, take the top ten moves in consideration (most likely those with the highest score). For each of these moves, generate the opponent's highest-scoring response based upon what we guess their rack to be. This involves calling the move generator 10, 000 times, but this should only take on the order of half a minute to do.

Now, for each of our ten best moves, we have the average score of our opponent's response. Using this average, we may choose which of those ten fares well both offensively and defensively. Unfortunately, that's half a minute of time not used efficiently. For example, after observing 50 opponent responses to move 9, what if the average score was dramatically lower than those of the other moves in consideration? We should try and remove it from the list as quick as possible so that we can perform more precise guesses for the remaining moves.

A fairly good heuristic to cull away nonideal moves quickly is to first compute the above averages in groups of 50 or 100 opponent responses at a time. For each of these, compute not only the average response score but the standard deviation as well. Using this standard deviation, compute a confidence interval (CI) of possible opponent response scores. Take the best move considered thusfar and consider the lower bound of its CI. Similarly, for the worst moves considered, calculate the upper bounds of their CI's. If the CI of a lesser move is disjoint from the CI of the best move under consideration, then we may ignore it.

An illustrative example is provided. Suppose the board is empty and our rack is ABDDRSV. What is the best play? We first decide to consider the top 22 moves, which are all possible placements of VAR, BRR, BARD, BRAD, DARB and DRAB. First we iterate for 100 random opponent racks

Average	SD	Position	Word	Best-case	Action
19.8	29.8	8h	DRAB	23.61	Keep
18.7	29.5	8g	DARB	22.48	Keep
18.7	32.0	8e	BARD	22.80	Keep
18.2	27.8	8f	DRAB	21.76	Keep
17.9	27.2	8f	BRAD	21.38	Keep
17.6	31.2	8e	DARB	21.59	Keep
17.6	28.5	8g	BARD	21.25	Keep
17.5	30.2	8h	VAR	21.37	Keep
16.9	27.6	8g	BRAD	20.43	Keep
16.8	28.3	8f	VAR	20.42	Keep
16.8	29.0	8g	DRAB	20.51	Keep
16.5	29.4	8h	DARB	20.26	Keep
16.3	27.7	8f	DARB	19.85	Keep
16.3	27.8	8f	BARD	19.86	Keep
16.0	29.6	8h	BRAD	19.79	Keep
15.5	29.2	8f	BRR	19.24	Keep
15.3	32.1	8e	BRAD	19.41	Keep
15.1	31.5	8e	DRAB	19.13	Keep
14.4	25.4	8g	BRR	17.65	Prune
13.4	30.0	8h	BARD	17.24	Prune
12.6	29.1	8h	BRR	16.32	Prune

Figure 5: Monte Carlo Simulation After N = 100 Iterations

and obtain the best average, which is move VAR through square 8g with an average opponent response of 22.7 and standard deviation of 33.3. The 80% CI states that we are 80% sure the worst this move will do is score 18.44. The other moves after 100 iterations is presented in the table in Figure 5 while moves after 200 iterations are presented in Figure 6.

Note that for the last three moves of Figure 5, the CI upper bound is lower than the CI lower bound for the best move. This is why they were pruned. After 200 iterations, the best move is again VAR 8g with a worst-case value of 22.90.

As can be seen, this reduction process can quickly remove certain plays out of consideration, especially when taking the global perspective using the Prairiefire supercomputer to compute these averages. Considering there are 256 processing elements, we may dispatch several requests for averages and standard deviations at a time. Given that there are an average of 16 turns per side in a Scrabble game, that leaves roughly 94 seconds of time to find the best move

Average	SD	Position	Word	Best-case	Action
22.2	29.2	8g	DARB	24.84	Keep
21.9	28.5	8f	DRAB	24.48	Keep
21.8	27.9	8f	BRAD	24.33	Keep
21.5	30.4	8h	DRAB	24.25	Keep
21.4	30.3	8f	VAR	24.14	Keep
21.2	28.7	8g	BARD	23.80	Keep
20.9	30.6	8h	VAR	23.67	Keep
20.4	28.4	8g	BRAD	22.97	Keep
19.9	31.5	8e	DARB	22.75	Prune
19.7	30.9	8h	BRAD	22.50	Prune
19.5	31.1	8e	BARD	22.31	Prune
19.0	28.2	8f	DARB	21.55	Prune
19.0	28.7	8g	DRAB	21.60	Prune
18.4	29.6	8h	DARB	21.08	Prune
17.9	28.0	8f	BARD	20.43	Prune
17.0	31.4	8e	BRAD	19.84	Prune
16.9	30.5	8e	DRAB	19.66	Prune
16.3	28.6	8f	BRR	18.89	Prune

Figure 6: Monte Carlo Simulation After N = 200 Iterations

Suppose the opponent plays QAT when the CUMQUAT bingo would have clearly been a far superior play. We would hence infer that the opponent does not have some or all of tiles CUUM.

To successfully compute probabilities of our opponent holding various tiles, the computer takes the tiles it knows its opponent had on the previous play, namely the ones that were just laid or exchanged. Next, the computer "fills in the gaps" by picking random tiles to fill the rest of the rack. If the best move on that rack is significantly better than the move played, a lower probability is assigned for the guessed tiles. After filling the gaps on the opponent's rack several thousands of times, the probabilities will converge sufficiently that they can be used for the next round of Monte Carlo simulations.

Towards the end of the game, before the bag is entirely empty, inference comes up with highly accurate rack probabilities. In addition, in the scenario when the opponent plays a large number of tiles such as four or five, the rate for accurately guessing tiles is surprisingly high, as there are few combinations of unseen tiles on the rack.

per turn. In that amount of time, Prairiefire can consider on the order of ten million different boards. However, in the above example, it only took 4100 to more the halve the number of considered plays, a feat Prairiefire can accomplish in approximately 1/25th of a second.

3.2 Tile Inference

Above, we see that the better our guesses for our opponent's rack, the better idea we have of their average response. Therefore, the better we can guess, the better we will do. To aide in this pursuit, we make one large assumption, primarily that our opponent is a good player. By this, we mean that they make very few of what we consider to be blunders.

3.3 Endgame Solvers

In the endgame, when the bag has become completely empty, Scrabble is a game of perfect information. Therefore, in comparitively little time, the computer can decide if it is possible to win, lose or tie in a given position. This is of course in great interest to a programmer, as Scrabble programs are notoriously bad in the endgame.

Supposing the bag has one tile and the opponent has N tiles, then an endgame solver must do N times as much work, trying all combinations of length N from a set of N + 1 tiles. If the bag has two tiles, the work is N + 2 choose N times as much, and so on.

While this is feasible for small numbers of tiles, it quickly

	Average	SD
Turns	26.004	2.716
Bingos	3.185	1.175
Scores		
Winning	447.055	43.305
Losing	371.632	38.851
First	415.923	55.749
Second	402.764	55.087

Figure 7: Per-game Averages

becomes unreasonable. However, it is a very important element of a competitive Scrabble program.

4. MATING SUPERCOMPUTERS AND SCRABBLE FOR STATISTICS

There are several uses for precomputed statistics about the average behavior in Scrabble games. If, after a billion games, there are some words of length 5 or 6 that have not yet been played, it's quite likely they're either impossible to play or have such a low probability of occurrence that a computer should completely ignore their existence. The computer would hence be able to reduce its vocabulary strategically to eke out slight performance enhancements, as move generation is affected by the lexicon size roughly logarithmically in time.

To test the robustness of Martin Landau, the author gen-

Word	Score
QuE(A)ZIER	365
CO(E)NZYME	356
QuEAZI(E)R	356
WH(E)EZIER	347
FRE(N)ZiLY	338
WHEEZ(I)Er	338
FRENZ(I)ED	329
F(r)OUZIER	320
MAGAZI(N)E	320
OBLIQU(E)D	320

Figure 8: Top 10 Highest Scoring Moves

Winning	Losing	Combined
826	568	1219
813	566	1203
799	564	1193
773	564	1186
772	563	1179
770	561	1178
766	561	1164
763	559	1159
762	555	1157
761	552	1155

Figure 9: Top 10 Highest Scores

erated and subsequently analyzed ten million games with it. On 200 processors of the Prairiefire supercomputer, this computation took under 90 minutes. The million games generated used the OSPD which, in its electronic incarnation, is composed of approximately 167, 000 words.

There were 17 unplayed 6 letter words, 7 unplayed 7's, and 14 unplayed 8's. All words 5 or less letters were played at least 32 times. Unplayed 6's included BUBBLE, KIOSKS, MOMMAS, PAZAZZ, TSKTSK, and others. While it is possible that these can be played, it is unlikely. To play PAZAZZ, it is required a player simultaneously have AAPZ and two blanks on his or her rack. Being as there are 2 blanks in a bag and only one Z, this is an unlikely occurrence. In addition, it is most likely that, with two blanks, there is a better move existing on the board. There were less unplayed 7's because of bingo bonuses, in which a player receives 50 extra points for playing all seven tiles. The unplayed 7's were GIGGING, KEBBUCK, KINKIN, MAMMOCK, PIPPINS, POPPLES and ZYZZYVA.

While the list of statistics can continue on forever, the reader is able to get an idea of what the author seeks. Even this short list of statistics has allowed us to make inferences upon playing characteristics. With more data, fast and strong heuristics can be developed to improve game-play.

Length of Word	Times Played
2	21120989
3	57258750
4	53288338
5	46856258
6	21866465
7	19798965
8	16743661
9	916656
10	145538
11	33036
12	6532
13	987
14	80

Figure 10: Word-Length Histogram

Word	Occurrences
QAT	1962165
XI	727552
QAID	634685
OX	552025
JO	531085
EX	499612
QUA	465100
AX	455195
IF	446699
AI	440012

Figure 11: Top 10 Played Words

5. CONCLUSIONS

The subject of computer Scrabble involves very fundamental aspects of Computer Science and is an exercise of great utility for a student learning the subject. Throughout the

course of the semester, the author has assembled a Scrabble program coined Martin Landau, which consists of 5200 lines of highly optimized and reused C ++ code, meeting most of the goals originally sought. Several discoveries have been made, such as new efficient methods for Scrabble lexicon storage, quick computation of rack management heuristics, and new methods for inference heuristics.

In the end, while some may consider Scrabble a game for rickety old housewives, the author, as well as many others, will consider it the most innovative game of its time.

6. REFERENCES

[1] A. W. Appel and G. J. Jacobson. The world's fastest Scrabble program. Commun. ACM, 31(5):572578, May 1988.

[2] J. Cosma and D. Jackson. Introducing monty plays scrabble. Scrabble Players News (June 1983), pages 710, June 1983.

[3] S. Gordon. A faster scrabble move generation algorithm. Software Practice and Experience, 24:219232, Feb. 1994.

[4] S. C. Shapiro. A scrabble crossword game playing program. Proceedings of the Sixth IJCAI, pages 797799, 1979.

[5] P. Turcan. A competitive scrabble program. SIGArt Newsletter, 80:104109, Apr. 1982.