A COMPARISON BETWEEN PROBABILISTIC SEARCH AND WEIGHTED HEURISTICS IN A GAME WITH INCOMPLETE INFORMATION

Steven Gordon
Department of Mathematics
East Carolina University
Greenville, NC 27858
EMAIL: magordon@ECUVAX.CIS.ECU.EDU

Abstract

Computing an effective strategy in games with incomplete information is much more difficult than in games where the status of every relevant factor is known. A weighted heuristic approach selects the move in a given position that maximizes a weighted sum of known factors, where the weights have been optimized over a large random sample of games. Probabilistic search is an alternative approach that generates a random set of scenarios, simulates how plausible moves perform under each scenario, and selects the move with the "best" overall performance. This paper compares the effectiveness of these approaches for the game of Scrabble.

Introduction

Computing an effective strategy in games with incomplete information is much more difficult than in games where the status of every relevant factor is known. One approach is to pick the move in the given position that maximizes a weighted sum of the factors that are known. The weights are those which yield the optimum performance over a large random sample of games. This approach will be called a weighted heuristic. An alternative approach is to pick a set of plausible candidate moves, generate a random sample of possible scenarios, simulate how each candidate move performs under each scenario, and play the move with the "best" overall performance. This statistical game tree approach will be called probabilistic search.

This paper compares the effectiveness of these two approaches in the game of Scrabble (in this paper Scrabble refers to the SCRABBLE® brand word game, a registered trade mark of Milton Bradley, a division of Hasbro, Inc.). While this paper will show that weighted heuristics are more reliable and much more efficient than probabilistic search, this paper will also argue that probabilistic search still needs further evaluation. Although weighted heuristics are effective, they are not responsive to individual positions, and therefore often make strategic blunders. The judicious use of probabilistic search could make a weighted heuristic into a more "intelligent" strategy.

Strategy in Scrabble

There are two basic skills involved in playing Scrabble, word-making and strategy. Although determining which words from a large lexicon can be played in a given position is the more difficult task for people, it is far easier to program than strategy. Appel and Jacobson [1] present a compact representation for a large lexicon and a fast algorithm for generating every possible move in a given Scrabble position. Gordon [8] presents a faster algorithm that uses a similar, but less compact representation (a 75,000 word lexicon took up 5 times as much space but moves were generated more than twice as fast). This paper addresses the more difficult problem of strategy.

A program (e.g., Appel and Jacobson's) that just plays the highest scoring legal move (i.e., the greedy evaluation function) has enough of a lexical advantage to beat most people, but would not fare well against good tournament players. North American Tournament Scrabble differs

from the popular version in that games are one–on–one, have a time limit of 25 minutes per side, and have a strict word challenge rule. When a play is challenged and is not in the standard dictionary [9], the play is removed, and the challenger gets to play next. Otherwise, the play stands and the challenger loses his/her turn. To the casual observer, the most apparent characteristic of tournament play is the many obscure words that are played (e.g., OE, QAT and ZEMSTVO). Nevertheless, strategy is a significant factor in competitive play. Otherwise, the greedy evaluation function should be able to win at least half of its games against good tournament players.

Attempts to model expert Scrabble strategy have had mixed success. A few commercially available programs do play at a competitive level, Tyler (copyrighted by Alan Frank), CrossWise (written by Jim Homan; copyrighted by Cygnus Cybernetics Corporation), and Maven (copyrighted by Brian Sheppard). These programs are proprietary (and CrossWise does not permit positions or racks to be dictated), so objective evaluation of their strategic heuristics is problematic. However, playing hundreds of games with the first two programs (the last program just became available recently) reveals meager, mechanical strategies. Nevertheless, these programs hold their own against good tournament players. While total command of the lexicon alone is not sufficient for expert play, it appears to be quite sufficient when combined with just a little strategy. There is every reason to believe that better strategic heuristics should enable these programs to consistently outplay the best human players.

Scrabble strategy is complex. Not knowing the opponent's tiles (except at the very end of the game) or what tiles will be drawn next would seem to preclude any kind of exhaustive analysis. When estimating which move is most likely to lead to victory, besides points scored, the most obvious factors to consider are:

1. Rack Leave: The future utility of the tiles left on the rack after a move.
2. Board Position: The future utility of the board after a move.

Commercial programs generally estimate these two factors in units of points. By adding these estimates for each move to its score, the tradeoffs between score, rack leave, and board position can be represented by a single number. The play with the best total evaluation is selected.

The following section presents a series of increasingly effective, rack leave evaluation functions implemented as weighted heuristics of the rack leave. Even more effective rack evaluation functions would seem feasible. However, analysis of Scrabble positions indicates that an optimal rack evaluation function would have to sometimes consider factors other than just the rack itself, such as the tiles remaining to be drawn and the tiles on the board that can be played through. This analysis also suggests that factoring in features of the board position could also increase effectiveness, but is problematic

Rack Evaluation Functions

The greedy evaluation function frequently chooses a move that uses a BLANK or S, when a more effective strategy would be to play a slightly lower scoring alternative that saves the valuable tile for a future move. A related defect is making a move that retains a Q or J when a more effective strategy would be to play a slightly lower scoring alternative that gets rid of the hard to use tile. A heuristic that addresses both defects is to evaluate a play by adding to its score an estimate of the future utility of the tiles left in the rack. Assigning large positive values to tiles like BLANK or S discourages their use except for a significantly higher score. Assigning large negative values to tiles like Q or J encourages their use even when somewhat higher scoring alternatives are available.

Ballard [3] presents the values listed as Rack Heuristic1 in Table 1 as a rule of thumb for human players. The sum of these values for the letters in a rack estimates its future utility. This also provides a heuristic for trading tiles (with loss of turn): trade in a subset of your tiles if the resulting rack leave has a higher utility than the sum of the score and rack evaluation of any other play. Table 2 shows that this weighted heuristic is an improvement on the greedy evaluation function (sacrifices, when the highest scoring move is not played, occur on about a third of its moves).

Accounting For Duplicate Tiles. Duplicate tiles reduce the number of unique combinations of tiles in a rack as well as the probability of playing all seven letters, and should therefore reduce the estimated utility of a rack. The contribution of

each letter in a rack would be evaluated more accurately by weighing both the utility of the letter by itself and the decrease in utility if the letter is duplicated. Table 1 lists values for Heuristic2. Instead of encoding separate penalties for triplication, quadruplication, etc., the duplication penalty is simply reapplied for each additional duplication. For instance, Heuristic1 values the rack IIISS at –1.5 + –1.5 + –1.5 + 7.5 + 7.5 = +10.5, whereas Heuristic2 values this rack at –0.5 + (–0.5 – 4.0) + (–0.5 – 4.0 – 4.0) + 7.5 + (7.5 – 4.0) = –2.5. Most experienced Scrabble players would agree that IIISS is usually an undesirable rack leave. Table 2 shows that Heuristic2 is significantly more effective against the greedy evaluation function than Heuristic1.

The final weights for Heuristic 2 were derived in the following manner:

1. Guess reasonable values for each weight.
2. Find the optimal value for each of the 49 weights by:
   a. Adjusting the weight by increments of 2.0 until performance in 1,000 games against the greedy algorithm is optimized.
   b. Repeating 2a with increments of 0.5 and 10,000 game tests.

This algorithm, somewhat reminiscent of neural net training algorithms [6], assumes that the original weights are close enough to optimal that each weight can be optimized independently.

The first version of this training procedure had just one phase for each weight with 0.5 increments and 1,000 game trials. Sometimes, one trial would win a few more games against the greedy algorithm than another trial, yet outscore the greedy algorithm by less points. Brian Sheppard [10] points out that cumulative point spread, being a far larger sample, is a more accurate statistic for comparing evaluation functions. Longer tests would be required to derive accurate winning percentages. He also notes that some duplication penalties in the first version of Rack Heuristic2 seemed too small.

Letter	Heuristic1	Heuristic2		Letter	Heuristic1	Heuristic2	
A	+0.5	+1.0	–3.0	B	–3.5	–3.5	–3.0
C	–0.5	–0.5	–3.5	D	–1.0	0.0	–2.5
E	+4.0	+4.0	–2.5	F	–3.0	–2.0	–2.0
G	–3.5	–2.0	–2.5	H	+0.5	+0.5	–3.5
I	–1.5	–0.5	–4.0	J	–2.5	–3.0	
K	–1.5	–2.5		L	–1.5	–1.0	–2.0
M	–0.5	–1.0	–2.0	N	0.0	+0.5	–2.5
O	–2.5	–1.5	–3.5	P	–1.5	–1.5	–2.5
Q	–11.5	–11.5		R	+1.0	+1.5	–3.5
S	+7.5	+7.5	–4.0	T	–1.0	0.0	–2.5
U	–4.5	–3.0	–3.0	V	–6.5	–5.5	–3.5
W	–4.0	–4.0	–4.5	X	+3.5	+3.5	
Y	–2.5	–2.0	–4.5	Z	+3.0	+2.0	
BLANK	+24.5	+24.5	–15.0				

TABLE 1. Weights for Rack Evaluation Heuristic1 and Heuristic2.

	Average Scores		Winning %	Moves	Sacrifices	Trades	Secs/ Move
Greedy vs. Greedy	388.8	388.8	50.0	253,563	0	0	0.489
RackH1 vs. Greedy	402.8	396.4	53.4	257,278	43,617	179	0.606
RackH2 vs. Greedy	411.9	388.6	59.5	251,743	41,834	207	0.624
RackH3 vs. Greedy	417.0	386.7	63.4	251,975	45,792	254	0.653

TABLE 2. Performance of Rack Heuristics in 10,000 Random Games on a VAX4300.

		CONSONANTS						
		0	1	2	3	4	5	6
V	0	0	0	−1	−2	−3	−4	−5
O	1	−1	1	1	0	−1	−2	
W	2	−2	0	2	2	1		
E	3	−3	−1	1	3			
L	4	−4	−2	0				
S	5	−5	−3					
	6	−6						

TABLE 3. Vowel–Consonant Mix Evaluation in Rack Evaluation Heuristic3.

The intuition that an optimal strategy should sometimes sacrifice point spread to increase winning percentage argues against ever using point spread as the sole measure of effectiveness. Instead, the weight setting procedure was modified as above. Using this two–phase procedure did increase some duplication penalties. Furthermore, point spread and winning percentage ceased to clash. Hindsight suggests that the smaller increments and shorter tests had sometimes mired the earlier training procedure in local maxima.

Vowel–Consonant Mix. The more even the mix of vowels and consonants in a rack the better. Words average about 60% consonants, so an extra consonant is far more useful than an extra vowel. The ad hoc function, VCMix = MIN(3V + 1, 3C) − L = MIN(3V + 1 − L, 2L − 3V) (where V = number of vowels, C = number of consonants and L = number of letters), was invented to show that augmenting Heuristic2 with a discrete function of the number of vowels and consonants in a rack leave could increase effectiveness. Table 3 lists the values this function returns for plausible numbers of vowels and consonants.. Table 2 shows that Rack Heuristic3, the sum of Rack Heuristic2 and VCMix, is more effective than Rack Heuristic2.

Future Work. A systematic search for the most effective weighted heuristic for vowel–consonant mix has not yet been attempted. One approach would be to search for the 6 weights that optimize the performance of the function $\mathrm{MIN}(W_1V + W_2, W_3C + W_4) + W_5V + W_6C$. A more appealing alternative would use the 28 values in Table 3 as independent weights. In retrospect, a more flexible implementation of VCMix would have been a simple look–up in Table 3.

Building a feed-forward neural net [6] to evaluate rack leaves would be an interesting exercise. Would the result be more effective? Would its weights or internal nodes correspond in any way to any of the weighted heuristics already developed? The most difficult problem in building such a neural net would be that each step in back propagation would require several long trials. Perhaps, Rack Heuristic3 could be used to train the network first, and then actual game–playing trials could be used to refine the weights further.

Limitations. Weighted heuristics can only be optimized with respect to overall performance. Consider choosing between the following three plays with the rack EENQRST:

1. QAT for 12 points through an A on the board, leaving EENRS on the rack.
2. QATS for 13 points through an A on the board, leaving EENR on the rack.
3. CENTERS for 38 points through a C on the board, leaving Q on the rack.

Rack Heuristic3 would evaluate the rack leaves for these plays at 17.0, 9.5, and −11.5, respectively, giving respective final evaluations of 29.0, 22.5, and 26.5 points. This just means that the heuristic weights that win the most games in the long run favor the first play. It does not mean that the first play is actually the best play in the current position. Many relevant factors have been ignored. On the rack leave side of the ledger, if both BLANKs and at least 2 Us have not been played yet and the game is more than halfway over, then the third play is probably the best choice.

Positional factors are much more complicated. If the opponent is likely to hold an S or a BLANK and the QAT play sets up a big crossplay (by simultaneously making the word QATS), then the second play may be best. The likelihood of the opponent holding an S or BLANK depends on how many of them have yet to be played and the opponent's recent plays. Recently trading in some but not all tiles implies a high likelihood. A low likelihood is implied if holding a BLANK or S would have allowed the opponent a much higher scoring alternative to a recent play.

However, the opening a move presents to the opponent cannot be considered in a vacuum. If there is already one good opening on the board

that you can not use profitably, then setting up a second opening guarantees one of them will be available next turn. On the other hand, if there is no good opening, setting one up is usually a losing proposition. Further complicating positional considerations is that you might be way ahead or way behind. If way ahead, eliminating openings will make it harder for your opponent to catch up, even if you end up scoring less. If way behind, then taking chances to open up opportunities for big scoring plays in the future may be your only hope for victory.

On the other hand, another way for the opponent to catch up is to play one tile at a time if you get stuck with an unplayable Q at the end of the game. Therefore, if ahead, it is advisable to play the Q as soon as possible. When far behind with this kind of rack, a common strategy is to trade in the Q. This makes the game last longer, giving you more turns to catch up, and may stick your opponent with the Q. See [4] for a more detailed example of how serious Scrabble players analyze other similarly complex positions.

Weighted heuristics have trouble dealing with how the priorities of these factors vary from situation to situation. A training algorithm can find relative weights for these factors that optimizes performance over the course of a large number of games. The resulting heuristic would be likely to pick the same move in the above example no matter the score, whether the opponent is likely to have an S or not, which letters remain to be drawn, and the quality of the openings already on the board. Although the strategy would win many games, it would intermittently make obvious errors in judgment. In other words, it would play like one of the strong commercial programs currently available: effectively, but not intelligently.

Probabilistic Search

Further complicating the heuristic by trying to extract the relevant features from positions on the 225 squares of a Scrabble board is unlikely to help much. One way to try to circumvent the limitations of weighted heuristics would be to simulate what might happen after each reasonable alternative. Moves that might be appropriate in most positions, but would be a mistake in the given position, should perform worse than a more appropriate move over a sufficiently large random sample of scenarios. Given the time constraint of 25 minutes per game (an average of 2 minutes per turn) in tournaments, it would be impractical to simulate what might happen after every possible move. So, weighted heuristics are still useful — for quickly paring down the number of candidate moves for simulation.

In order to test the effectiveness of probabilistic search, a program was written that chose the C best candidate moves according to Rack Heuristic3 and generated S random scenarios. Each scenario consists of a random rack for the opponent and a random draw to replace the letters used by a candidate move. Each candidate move is played out in each scenario by playing the candidate move, then having the opponent play its "best" move, and then having the first player play its "best" comeback.

The "best" move for each player could be determined by any of the evaluation functions already developed. One could argue that Rack Heuristic3 should be used because of its effectiveness at winning games. However, it might choose a lower scoring play in the simulation. This choice might be objectively correct in that it might pay off handsomely a few turns later, but it would unfairly decrease the simulated value of a candidate move. So, if a heuristic is used, the heuristic's evaluation of a move must be used rather than the raw score in comparing the outcome of each scenario in the simulation.

A simpler alternative is to use the greedy evaluation function. Then the outcome of each scenario is just the resulting point spread (= its score – the opponent's score + the comeback score).

Preliminary results. Table 4 presents the results achieved by this probabilistic search program against the greedy evaluation program. Each test lasted only 100 games since probabilistic search takes so much longer than simple evaluation functions (they would have taken over twice as long without the improvement in move generation time in [8]). Two different sets of 100 games were used. Two different methods, greedy and Rack Heuristic3, were used to choose moves within the simulation.

The rows with 0 candidates and 0 solutions correspond to just running Rack Heuristic3 on the set of games without any simulations. On both test sets, the weighted heuristic is slightly more

Simulation Method	Number of Candidates	Number of Scenarios	Winning Margin	Winning %	Rank By RH3	Secs/ Move
Set1:						
Greedy	2	64	+33.7	68.0	1.26	178.64
Greedy	2	32	+32.6	67.5	1.28	91.21
RH3	2	32	+4.6	57.0	1.26	94.38
Greedy	4	8	+28.1	59.0	1.70	47.56
Greedy	4	16	+22.9	62.0	1.60	91.73
RH3	4	16	+1.9	46.0	1.69	95.07
Greedy	8	4	+15.6	65.0	2.51	49.73
RH3	8	8	+6.5	56.5	2.47	94.18
Greedy	8	8	+26.3	66.5	2.43	92.48
	0	0	+47.3	69.5	1.00	0.72
Set2:						
Greedy	2	64	+20.4	58.0	1.21	177.37
Greedy	2	32	+17.2	55.0	1.23	90.82
RH3	2	32	+16.8	57.5	1.22	94.44
Greedy	4	8	+8.6	53.5	1.68	47.79
Greedy	4	16	+24.6	58.0	1.62	92.12
RH3	4	16	+20.1	62.0	1.66	94.85
Greedy	8	4	+1.3	55.0	2.43	48.63
Greedy	8	8	+12.4	59.0	2.22	91.84
RH3	8	8	-9.0	45.0	2.48	93.91
	0	0	+34.9	63.5	1.00	0.70

TABLE 4. Performance of Probabilistic Search in Two Sets of 100 Random Games.

effective than even the most effective of the probabilistic searches.

The rank statistics in Table 4 are the average rank of the play selected by probabilistic search among the candidates with respect to the number of points they scored and with respect to their evaluation by Rack Heuristic3. Doubling the number of scenarios simulated only reduces both average ranks slightly. This indicates that Rack Heuristic3 is a good predictor of the average effectiveness of plays. Surprisingly, doubling the number of scenarios did not always improve winning percentage.

Other discrepancies in effectiveness are also apparent. Not only did doubling the number of candidates sometimes increase and somtimes decrease effectiveness, but the effectiveness varied quite a bit between greedy simulation and heuristic simulation. Another discrepancy is between the size of the scoring margin and the winning percentage. These discrepancies all call into question the reliability of probabilistic search with a small number of scenarios. However, increasing the number of scenarios would be impractical if probabilistic search was going to be performed on each and every move.

Future Work. An investigation into when probabilistic search is effective and when it is unnecessary could be fruitful. Probabilistic search may well prove ineffective in the first half of games. Another time that a search may be unnecessary is when the weighted heuristic has found a single superior move. It will likely prove especially effective near the end of the game when scenarios are apt to be more accurate.

Generating 4-ply scenarios could make probabilistic search more effective, although doubling the number of scenarios would seem generally more effective than doubling the number of plies. Using the opponent's recent plays as a guide in generating more realistic scenarios is another way to try to improve effectiveness.

Another idea worth exploring is that average performance over the set of scenarios may be the appropriate measure only in closely contested games. When ahead by 100 or move points, winning percentage could be increased with a

more conservative approach that makes it even harder for the opposition to catch up. Under this proposal, the "best" move might be the one that allows the opposition to gain 50 or more points in the fewest scenarios. A move that gains 30 points on average, but allows the opposition to make a huge play in some scenarios might be passed up in favor of a lower scoring play that does not give the opposition as much of a chance for a big comeback.

An analogous strategy when behind by 100 or more points would be to choose the move that gains 50 or more points on the opposition in the most scenarios. A move that gains 30 points on average, but makes a big play next move unlikely might be passed up in favor of a lower scoring play that increases the chance of a big play next turn. These two strategies could be expected to translate into a slightly higher winning percentage with a somewhat smaller average margin of victory.

Conclusion

The main advantage that probabilistic search should have over weighted heuristics is that it responds to specific situations. It should reject a routine move when a less standard alternative performs better in simulations because it is more appropriate for the given position.

Unfortunately, the computational inefficiency of probabilistic search limits the number of candidates and scenarios it can simulate in practice. This undermines not only its potential effectiveness as a Scrabble strategy, but also the investigation into its potential effectiveness. Further investigation into the conditions under which probabilistic search is effective and how many scenarios are required for reliability could yield a practical way to increase the intelligence of the current effective, but unintelligent Scrabble strategies based on weighted heuristics.

The tradeoff between the inefficiency and the potential effectiveness of probabilistic search in Scrabble parallels concerns about the real-time use of simulations in other games with incomplete information, such as Bridge [7].

On the other hand, theoretical research into extending game tree search techniques to games with incomplete information, such as [2] and [5], is mainly concerned with tractability. While probabilistic search is inefficient compared to weighted heuristics, it is by no means intractable. The practical real-time application of theoretical research into extending game tree techniques to games with incomplete information will have to overcome problems similar to those addressed in this paper.

References

1. A. Appel and G. Jacobson, 'The world's fastest Scrabble program', *Commun. ACM*, 31,(5), 572–578, 585 (May 1988).
2. B. Ballard, 'The *-minmax search procedure for trees containing chance nodes', *AI*, 21, (1,2), 327-350 (March 1983).
3. N. Ballard, 'Renaissance of Scrabble theory 2', *Games Medleys*, 17, 4–7 (May 1992).
4. N. Ballard, 'The faint of heart are missing the boat', *Games Medleys*, 24, 3–6 (Jan. 1993).
5. J. Blair, D. Mutchler, and C. Liu, 'Heuristic search in one-player games with hidden information', Technical Report No. CS–92–161, Department of Computer Science, University of Tennessee (June 1992).
6. P. S. Churchland and T. Sejnowski, *The Computational Brain*, MIT Press 1992, 17–48.
7. B. Gamback, M. Rayner, and B. Pell, 'Pragmatic reasoning in Bridge', Technical Report No. 299, University of Cambridge, Computer Laboratory (April 1993).
8. S. Gordon, 'A faster Scrabble move generation algorithm', Submitted for publication (March 1993).
9. Milton Bradley Company, *The Official Scrabble Players Dictionary, 2nd ed.*, Merriam–Webster Inc., Springfield, MA 1990.
10. B. Sheppard, personal communication (April 1993).